dBASE Plus 9 LR

Table of Contents

Language Reference 1
Language Definition 1
Language Syntax 17
Operators and Symbols 20
IDE Language Elements 33
Application Shell 62
Core Language 117
Syntax 143
Data Objects 160
Form Objects 272
Report Objects 480
Files and Operating System 516
Xbase 558
Local SQL 659
Arrays 673
Date and Time 713
Math / Money 742
Strings 766
Text Streaming 793
ActiveX 820
Bitwise 842
Miscellaneous Language Elements 847
Extending dBASE Plus 864
Preprocessor 883
BDE Limits 893
Index 897

Language Reference
Language Definition

Language definition

Topic group Related topics

dBL is a dynamic object-oriented programming language. It features dozens of built-in classes that represent forms, visual components, reports, and databases in an advanced integrated development environment with Two-Way Tool designers.

These topics define the language elements in dBL. After a brief overview of basic language attributes, which is geared toward those with previous programming experience, the language is described from its most fundamental elements, data types, to the most general.

Basic attributes

Topic group Related topics

If you're familiar with another programming language, knowing the following attributes will help orient you to dBL. If dBL is your first programming language, you may not recognize some of the terminology below. Keep the rules in mind; the terminology will be explained later in this series of topics.

dBL is not case-sensitive.

Although language elements are capitalized using certain conventions in the *dBL Language Reference*, you are not required to follow these conventions.

Rules of thumb for how things are capitalized are listed in Syntax conventions. You are encouraged to follow these rules when you create your own names for variables and properties.

dBL is line-oriented.

By default, there is one line per statement, and one statement per line. You may use the semicolon (;) to continue a statement on the next line, or to combine multiple statements on the same line.

Most structural language elements use keyword pairs.

Most structual language elements start with a specific keyword, and end with a paired keyword. The ending keyword is usually the word starting keyword preceded by the word END; for example IF/ENDIF, CLASS/ENDCLASS, and TRY/ENDTRY.

Literal strings are delimited by single quotes, double quotes, or square brackets.

dBL is weakly typed with automatic type conversion.

You don't have to declare a variable before you use it. You can change the type of a variable at any time.

dBASE Plus's object model supports dynamic subclassing.

Dynamic subclassing allows you to add new properties on-the-fly, properties that were not declared in the class structure.

Command Line Switches

Topic group

When running dBASE Plus.exe, Plusrun.exe or an Application .exe built by dBASE, there are several Command Line Switches which can be used to override certain defaults.

Switches
The command line switch, -c<.ini file path>, specifies an alternate .ini file to be used in place of the default.

For example:
"c:\program files\dbase\plus\bin\plus.exe" -cC:\iniPath //open dBASE Plus using C:\iniPath\Plus.ini

The command line switch, -v0, specifies that _app.useUACPaths is to be set to 'False'
The command line switch, -v1, specifies that _app.useUACPaths is to be set to 'True'

For example:
//open App.exe with it's _app.useUACPaths = false
"c:\program files\YourApp\App.exe" -v0
//open dbase Plus.exe with it's _app.useUACPaths = true
"c:\program files\dBASE\Plus\Bin\Plus.exe" -v1

Data types

Topic group Related topics

Data is both the means and the end for both programming and databases. Because *dBASE Plus* is designed to manipulate databases, there are three categories of data types:

- Simple data types common to both the language and databases
- Database-specific data types
- Data types used in programming

Simple data types

Topic group Related topics

There are five simple data types common to both *dBASE Plus* and databases:

- String
- Numeric
- Logical or boolean
- Date
- Null

Keep in mind that different table formats support different data types to varying degrees.

For each of these data types, there is a way to designate a value of that type in dBL code. This is known as the literal representation.

String data

Topic group Related topics

A string is composed of zero or more characters: letters, digits, spaces, or special symbols. A string with no characters is called an empty string or a null string (not to be confused with the null data type).

The maximum number of characters allowed in a string depends on where that string is stored. In *dBASE Plus*, the maximum is approximately 1 billion characters, if you have enough virtual memory. For DBF (dBASE¨) tables, you may store 254 characters in a character field and an unlimited number in a memo field. For DB (Paradox) tables, the limit is 255 characters in an alpha field, and no limit with memo fields. Different database servers on different platforms each have their own limits.

Literal character strings must be enclosed in matching single or double quotation marks, or square brackets, as shown in the following examples:

```
'text'
"text"
[text]
```

A literal null string, or empty string, is indicated by two matching quotation marks or a set of square brackets with nothing in between.

Numeric data

Topic group Related topics

dBL supports a single numeric data type. It does not distinguish between integers and non-integers, which are also referred to as floating-point numbers. Table formats vary in the types of numbers they store. Some support short (16-bit) and long (32-bit) integers or currency in addition to a numeric format. When these numbers are read into *dBASE Plus*, they are all treated as plain numbers. When numbers are stored into tables, they are automatically truncated to fit the table format.

In dBL, a numeric literal may contain a fractional portion, or be multiplied by a power of 10. The following are all valid numeric literals:

42
5e7
.315
19e+4
4.6
8.306E–2

As the examples show, the "E" to designate a power of 10 may be uppercase or lowercase, and you may include a plus sign to indicate a positive power of 10 even though it is unnecessary.

In addition to decimal literals, you may use octal (base 8) or hexadecimal (base 16) literal integers. If an integer starts with a zero (0), it is assumed to be octal, with digits from 0 to 7. If it starts with 0x or 0X, it is hexadecimal, with the digits from 0 to 9 and the letters A to F, uppercase or lowercase. For example,

Literal	Base	Decimal value
031	Octal	25
0x64	Hexadecimal	100

Logical data

Topic group Related topics

A logical, or boolean, value can be one of three things: true, false or null. These logical values are expressed literally in dBL by the keywords *true, false* and null.

For compatibility with earlier versions of dBASE, you may also express *true* as .T. or .Y., and *false* as .F. or .N.

Date data

Topic group Related topics

dBASE Plus features native support for dates, including date arithmetic. To specify a literal date, enclose the date in curly braces. The order of the day, month, and year depends on the current setting of SET DATE, which derives its default setting from the Regional Settings in the Windows Control Panel. For example, if SET DATE is MDY (month/day/year), then the literal date:

```
{10/02/97}
```

is October 2nd, 1997. The way *dBASE Plus* handles two-digit years depends on the setting of SET EPOCH. The default is to interpret two-digit years between 50 and 99 as a year in the 1900s. Two digit years between 00 and 49 will be interpreted as a year in the 2000s. Curly braces with nothing between them represent a special date value, known as a blank date.

Null values

Topic group Related topics

dBASE Plus supports a special value represented by the keyword null. It is its own data type, and is used to indicate a nonexistent or undefined value. A null value is different from a blank or zero value; null is the absence of a value.

The new DBF7 (dBASE) table type support nulls, as do most other tables, including DB (Paradox). Older DBF formats do not. A null value in a field would indicate that no data has been entered into the field, like in a new row, or that the field has been emptied on purpose. In certain summary operations, null fields are ignored. For example, if you are averaging a numeric field, rows with a null value in the field are ignored. If instead a null value was considered to be zero or some other value, it would affect the average.

null is also used in *dBASE Plus* to indicate an empty function pointer, a property or variable that is supposed to refer to a function, but doesn't contain anything.

Database-specific data types

Topic group Related topics

There are a number of data types supported by different databases that do not have a direct equivalent in *dBASE Plus*. The following list is not exhaustive; a new or upgraded table format may introduce new types. In any case, the type is represented by the closest matching *dBASE Plus* data type, with the string type being the catchall, since all data can be represented as a bunch of bytes.

The common database-specific types are:

Memo

Memo data

Topic group Related topics

As far as *dBASE Plus* is concerned, a memo is just a character string; potentially a very long one. For tables, it is important to distinguish between a character field, which is of fixed and usually small size, and a memo field, which is unlimited in size. For example, a character field might contain the title of a court decision, and the memo field contain the actual text of that court decision.

Binary and OLE data

Topic group Related topics

Binary and OLE data are similar to memos, except that they are usually meant to be modified by external programs, not *dBASE Plus*. For example, a binary field might contain a graphic bitmap, which *dBASE Plus* can display, but you cannot edit the bitmap with *dBASE Plus*.

Programming data types

Topic group Related topics

There are three data types used specifically for programming:

- Object reference
- Function pointer
- Codeblock

These types are explained later, in the context in which they are used.

Operators and symbols

Topic group Related topics

An operator is a symbol, set of symbols, or keyword that performs an operation on data. dBL provides many types of operators, used throughout the language, in the following categories:

Operator category	Operator symbols
Assignment	`= := += -= *= /= %=`
Comparison	`= == <> # > < >= <= $`
String concatenation	`+ -`
Numeric	`+ - * / %` `^ ** ++ --`
Logical	`AND OR NOT`
Object	`. [] NEW ::`
Call, Indirection	`( )`

Alias	`->`
Macro	`&`

All operators require either one or two arguments, called *operands*. Those that require a single operand are called *unary operators*; those requiring two operands are called *binary operators*. For example, the logical NOT operator is a unary operator:

```
not endOfSet
```

The (*) is the binary operator for multiplication, for example,

```
59 * 436
```

If you see a symbol in dBL code, it's probably an operator, but not all symbols are operators. For example, quote marks are used to denote literal strings, but are not operators, since they do not act upon data—they are part of the representation of a data type.

Another common symbol is the end-of-line comment symbol, a double slash. It and everything on the line after it are ignored by *dBASE Plus*. For example,

```
calcAverages( ) // Call the function named calcAverages
```

All operators and symbols are described in full in the Operators and Symbols section of this Help file.

Names

Topic group Related topics

Names are given to variables, fields in work areas, properties, events, methods, functions, and classes. The following rules are the naming conventions in dBL:

A name begins with an underscore or letter, and contains any combination of underscores, letters, spaces, or digits.
If the name contains spaces, it must be enclosed in colons.
The letters may be uppercase or lowercase. dBL is not case-sensitive.

With dBL, only the first 32 characters in a name are significant. There can be more than 32, but the extra characters are ignored. For example, the following two names are considered to be the same:

```
theFirst_32_CharactersAreTheSameButTheRestArent
theFirst_32_CharactersAreTheSameAndTheRestDontMatter
```

The following are some examples of valid names:

```
x
:First name:
DbException
Form
messages1_onOpen
```

Filename skeletons

Topic group

A Filename skeleton is a character string used as a template to search for matching filenames. It consists of an optional directory path followed by a backslash, \, followed by a filename template. The template can contain a mix of required characters plus the wildcard, or "placeholder", characters ? and *.

```
C:\MyFile.txt
C:\MyFile*.*
```

```
C:\??File.txt
```

A wildcard character can be used to represent any character that occupies the same position in the filename. A question mark, ?, will match any single character. An asterisk, *, will match any group of characters.

Any other characters in the template must exactly match the filenames' character, and its' position, to be considered a match.

For example:

.	All filenames are matches. ***.*** Specifies that any group of characters to the left of the period are valid matches. ***.**** Specifies that any group of characters to the right of the period are valid matches.
*.txt	All files with an extension of **.txt** are matches.
MyTable.*	All files, regardless of their extension, whose base name is **MyTable** are matches.
A?c.wf?	All files starting with an **A**, followed by any character, followed by a **c.wf** and ending in any character are matches.
C:\myfolder*.dbf	All files in folder **C:\myfolder** with a **.dbf** extension are matches.

Expressions

Topic group Related topics

An expression is anything that results in a value. Expressions are built from literal data, names, and operators.

Basic expressions

Topic group Related topics

The simplest expression is a single literal data value; for example,

```
6 // The number 6
"eloign" // The string "eloign"
```

You can use operators to join multiple literals; for example,

```
6 + 456 * 3 // The number 1374
"sep" + "a" + "rat" + "e" // The string "separate"
```

To see the value of an expression in the Command window, precede the expression with the ? symbol:

```
? 6 + 456 * 3 // Displays 1374
```

Variables

Topic group Related topics

Variables are named locations in memory where you store data values: strings, numbers, logical values, dates, nulls, object references, function pointers, and codeblocks. You assign each of these values a name so that you can later retrieve them or change them.

You can use these values to store user input, perform calculations, do comparisons, define values that are used as parameters for other statements, and much more.

Assigning variables

Topic group Related topics

Before a variable can be used, a value must be assigned to it. Use a single equal sign to assign an expression to a variable; for example,

```
alpha = 6 + 456 * 3 // alpha now contains 1374
```

If the variable does not exist, it is created. There are special assignment operators that will assign to existing variables only, and others that combine an arithmetic operation and an assignment.

Using variables and field names in expressions

Topic group Related topics

When a variable is not the target (on the left side) of an assignment operator, its value is retrieved. For example, type the following lines in the Command window, without the comments:

```
alpha = 6 // Assigns 6 to alpha
beta = alpha * 4 // Assigns values of alpha (6) times 4 to beta
? beta // Displays 24
```

In the same way, when the name of a field in a work area is used in an expression, its value for the current record is retrieved. (Note that assignment operators do not work on fields in work areas; you must use the REPLACE command.) Continuing the previous example:

```
use FISH // Open Fish table in current work area
? Name // Display value of Name field in first record
? :Length CM: // Display value of Length CM field in first record
// Colons required around field name because it contains spaces
? :Length CM: * beta // Display value of field multiplied by variable
```

For information on referencing fields in different work areas and resolving name conflicts between variables and field names, see Alias operator.

Type conversion

Topic group Related topics

When combining data of two different types with operators, they must be converted to a common type. If the type conversion does not occur automatically, it must be done explicitly.

Automatic type conversion

Topic group Related topics

dBASE Plus features automatic type conversion between its simple data types. When a particular type is expected, either as part of an operation or because a property is of a particular type, automatic conversion may occur. In particular, both numbers and logical values are converted into strings, as shown in the following examples:

```
"There are " + 6 * 2 + " in a dozen" // The string "There are 12 in a dozen"
"" + 4 // The string "4"
"2 + 2 equals 5 is " + ( 2 + 2 == 5 ) // The string "2 + 2 equals 5 is false"
```

As shown above, to convert a number into a string, simply add the number to an empty string. Be careful, though; the following expression doesn't work as you might expect:

```
"The answer is " + 12 + 1 // The string "The answer is 121"
```

The number 12 is converted to a string and concatenated, then the number 1 is converted and concatenated, yielding "121". To concatenate the sum of 12 plus 1, use parentheses to force the addition to be performed first:

```
"The answer is " + (12 + 1) // The string "The answer is 13"
```

Explicit type conversion

Topic group Related topics

In addition to automatic type conversion, there are a number of functions to convert from one type to another:

String to number: use the VAL( ) function
Number to formatted string: use the STR( ) function
Date to string: use the DTOC( ) function
String to date: use the CTOD( ) function

Arrays

Topic group Related topics

dBASE Plus supports a rich set of array classes. An array is an n-dimensional list of values stored in memory. Each entry in the array is called an element, and each element in an array can be treated like a variable.

To create an array, you can use the object syntax detailed in Array objects, but for a one-dimensional array, you can also use the literal array syntax.

Literal arrays

Topic group Related topics

A literal array declares and populates an array in a single expression. For example,

```
aTest = { 4, "yclept", true }
```

creates an array with three elements:

The number 4
The string "yclept"
The logical value *true*

and assigns it to the variable aTest. The three elements are enclosed in curly braces (the same curly braces used for dates) and separated by commas.

Array elements are referenced with the index operator, the square brackets ([ ]). Elements are numbered from one. For example, the third element is element number 3:

```
? aTest[ 3 ] // Displays true
```

You can assign a new value directly to an element, just like a variable:

```
aTest[ 3 ] = false // Element now contains false
```

Complex expressions

Topic group Related topics

The following is an example of a complex expression that uses multiple names, operators, and literal data. It is preceded by a question mark so that when it's typed into the Command window, it displays the resulting value:

```
? {"1st","2nd","3rd","4th"}[ ceiling( month( date( )) / 3 ) ] + " quarter"
```

Except for the question mark, the entire line is a single complex expression, made up of many smaller basic expressions. The expression is evaluated as follows:

1. A literal array of literal strings is enclosed in braces, separated by commas. The strings are enclosed in double quotation marks.
    - The resulting array is referenced using the square brackets as the index operator. Inside the square brackets is a numeric expression.
    - The numeric expression uses nested functions, which are evaluated from the inside out. First, the DATE( ) function returns the current date. The MONTH( ) function returns the month of the current date.
    - The month is divided by the number 3, then the CEILING( ) function rounds the number up to the nearest integer.
    - The string containing the ordinal number for the calendar quarter that corresponds to the month of the current date is extracted from the array, which is then added to the literal string "quarter".

The value of this complex expression is a string like "4th quarter".

Statements

Topic group Related topics

A statement is an instruction that directs *dBASE Plus* to perfom a single action. This action may be simple or it may be complex, causing other actions to occur. You may type and execute individual statements in the Command window.

Basic statements

Topic group Related topics

There are four types of basic statements:

1. dBL commands: These commands make up a significant portion of the entries in the *dBL Language Reference*. For example:

```
clear // Clears the Command window
erase TEMP.TXT // Erases a file on the disk
build from FISHBASE // Creates an executable
? time( ) // Displays the current time
```

2. Assignment statements: A statement may include only one assignment operator, although the value assigned may be a very complex expression. For example:

```
clear = 14 // Assign 14 to variable named clear
f = new Form( ) // NEW and call operator on class name Form, assigned to
variable f
```

Note that the first example uses the word "clear", but because the syntax of the statement a variable is created instead of executing the command. While creating variables with the same name as a command keyword is allowed, it is strongly discouraged.

3. dBL expressions: An expression is a valid statement. If the expression evaluates to a number, it is equivalent to a GO command. For example:

```
6 // Goto record 6
3 + 4 // Goto record 7
date( ) // Get today's date and throw it away
f.open( ) // Call object f's open( ) method
```

4. Embedded SQL statements: *dBASE Plus* features native support for SQL statements. You may type an SQL statement in the Command window, or include them in programs. If the command results in an answer table, that table is opened in the current work area. For example:

```
select * from FISH // Open FISH table in current work area
```

Control statements

Topic group Related topics

dBASE Plus supports a number of control statements that can affect the execution of other statements. Control statements fall into the following categories:

Conditional execution
IF
DO CASE
Looping
FOR
DO WHILE
DO...UNTIL
Object manipulation
WITH
Exception handling
TRY

These control statements are fully documented in the Core language topic series.

Functions and codeblocks

Topic group Related topics

In addition to the built-in functions, you may create your own. A function is a code module—a set of statements—to which a name is assigned. The statements can be called by the function name as often as needed. Functions also provide a mechanism whereby the function can take one or more parameters that are acted upon by the function.

A function is called by following the function name with a set of parentheses, which act as the call operator. When discussing a function, the parentheses are included to help distinguish functions from other language elements like variables.

For example, the function LDoM( ) takes a date parameter dArg and returns the last day of the month of that date.

```
function LDoM( dArg )
local dNextMonth
```

```
dNextMonth = dArg - date( dArg ) + 45 // Day in the middle of next month
return dNextMonth - day( dNextMonth )
```

Functions are identified by the keyword FUNCTION in a program file; they cannot be typed into the Command window. While many functions use RETURN to return a value, they are not required to do so.

Function pointers

Topic group Related topics

The name of a function that you create is actually a pointer to that function. Applying the call operator (a pair of open and closed parenthesis) to a function pointer calls that function. (Built-in functions work differently; there is no function pointer.)

Function pointers are a distinct data type, and can be assigned to other variables or passed as parameters. The function can then be called through that function pointer variable.

Function pointers enable you to assign a particular function to a variable or property. The decision can be made up front and changed as needed. Then that function can be called as needed, without having to decide which function to call every time.

Codeblocks

Topic group Related topics

While a function pointer points to a function defined in a program, a codeblock is compiled code that can be stored in a variable or property. Codeblocks do not require a separate program; they actually contain code. Codeblocks are another distinct data type that can be stored in variables or properties and passed as parameters, just like function pointers.

Codeblocks are called with the same call operator that functions use, and may receive parameters.

There are two types of codeblocks:

1. Expression codeblocks
    - Statement codeblocks

Expression codeblocks return the value of a single expression. Statement codeblocks act like functions; they contain one or more statements, and may return a value.

In terms of syntax, both kinds of codeblocks are enclosed in curly braces ({ }) and

Cannot span multiple lines.
Must start with either two pipe characters (||) or a semicolon (;)
If ; it must be a statement codeblock with no parameters
If || it may be either an expression or statement codeblock

The || are used for parameters to the codeblock, which are placed between the two pipe characters. They may also have nothing in-between, meaning no parameters for either an expression or statement codeblock.

Parameters inside the ||, if any, are separated by commas.

For an expression codeblock, the || must be followed by one and only one expression, with no ; These are valid expression codeblocks:

```
{|| false}
{|| date( )}
{|x| x * x}
```

Otherwise, it is a statement codeblock. A statement codeblock may begin with || (again, with or without parameters in-between).

Each statement in a statement codeblock must be preceded by a ; symbol. These are valid statement codeblocks (the first two are functionally the same):

```
{; clear}
{||; clear}
{|x|; ? x}
{|x|; clear; ? x}
```

You may use a RETURN inside a statement codeblock, just like with any other function. (A RETURN is implied with an expression codeblock.) For example,

```
{|n|; for i=2 to sqrt(n); if n % i == 0; return false; endif; endfor; return true}
```

Because codeblocks don't rely on functions in programs, you can create them in the Command window. For example,

```
square = {|x| x * x} // Expression codeblock
? square( 4 ) // Displays 16
// A statement codeblock that returns true if a number is prime
p = {|n|; for i=2 to sqrt(n); if n % i == 0; return false; endif; endfor; return
true}
? p( 23 ) // Displays true
? p( 25 ) // Displays false
```

As mentioned previously, curly braces are also used for literal dates and literal arrays. Compare the following:

```
{10} // A literal array containing one element with the value 10
{10/5} // A literal array containing one element with the value 2
{10/5/97} // A literal date
{||10/5} // An expression codeblock that returns 2
```

Codeblocks vs. functions

Topic group Related topics

A codeblock is a convenient way to create a small anonymous function and assign it directly to a variable or property. The code is physically close to its usage and easy to see. In contrast, a function pointer refers to a function defined elsewhere, perhaps much later in the same program file, or in a different program file.

Functions are easier to maintain. Their syntax is not cramped like codeblocks, and it's easier to include readable comments in the code. In a class definition, all FUNCTION definitions are all together at the bottom. Codeblocks are scattered throughout the constructor. If you want to run the same code from multiple locations, using function pointers that point to the same function means that changing the code requires changing the function once; multiple codeblocks would require changing each codeblock individually.

You can create a codeblock at runtime by constructing a string that looks like a codeblock and using the macro operator to evaluate it.

Objects and classes

Topic group Related topics

An object is a collection of properties. Each of these properties has a name. These properties may be simple data values, such as numbers or strings, or references to code, such as function

pointers and codeblocks. A property that references code is called a method. A method that is called by *dBASE Plus* in response to a user action is called an event.

Objects are used to represent abstract programming constructs, like arrays and files, and visual components, like buttons and forms. All objects are initially based on a class, which acts as a template for the object. For example, the PushButton class contains properties that describe the position of the button, the text that appears on the button, and what the button should do when it is clicked. All these properties have default values. Individual button objects are instances of the PushButton class that have different values for the properties of the button.

dBASE Plus contains many built-in, or stock, classes, which are documented throughout the *dBL Language Reference*. You can extend these stock classes or build your own from scratch with a new CLASS definition.

While the class acts as a formal definition of an object, you can always add properties as needed. This is called dynamic subclassing.

Dynamic subclassing

Topic group Related topics

To demonstrate dynamic subclassing, start with the simplest object: an instance of the Object class. The Object class has no properties. To create an object, use the NEW operator, along with the class name and the call operator, which would include any parameters for the class (none are used for the Object class).

```
obj = new Object( )
```

This statement creates a new instance of the Object class and assigns an object reference to the variable obj. Unlike variables that contain simple data types, which actually contain the value, an object reference variable contains only a reference to the object, not the object itself. This also means that making a copy of the variable:

```
copy = obj
```

does not duplicate the object. Instead, you now have two variables that refer to the same object.

To assign values to properties, use the dot operator. For example,

```
obj.name = "triangle"
obj.sides = 3
obj.length = 4
```

If the property does not exist, it is added; otherwise, the value of the property is simply reassigned. This behavior can cause simple bugs in your programs. If you mistype a property name during an assignment, for example,

```
obj.wides = 4 // should be s, not w
```

a new property is created instead of changing the value of the existing property you intended. To catch these kinds of problems, use the assignment-only := operator when you know you are not initializing a property or variable. If you attempt to assign a value to a property or variable that does not exist, an error occurs instead of creating the property or variable. For example:

```
obj.wides := 4 // Error if wides property does not already exist
```

Methods

Topic group Related topics

A method is a function or codeblock assigned to a property. The method is then called through the object via the dot and call operators. Continuing the example above:

```
obj.perimeter = {|| this.sides * this.length}
? obj.perimeter( ) // Displays 12
```

As you may have deduced by now, the object referred to by the variable obj represents a regular polygon. The perimeter of such a polygon is the product of the length of each side and the number of sides.

The reference *this* is used to access these values. In the method of an object, the reference *this* always refers to the object that called the method. By using *this*, you can write code that can be shared by different objects, and even different classes, as long as the property names are the same.

A simple class

Topic group Related topics

Here is a class representing the polygon:

```
class RegPolygon
this.sides = 3 // Default number of sides
this.length = 1 // and default length
 function perimeter( )
 return this.sides * this.length
endclass
```

The top of the CLASS definition, up to the first FUNCTION, is called the class constructor, which is executed when an instance of the class is created. In the constructor, the reference *this* refers to the object being created. The sides and length properties are added, just as they were before.

The function in the class definition is considered a method, and the object automatically has a property with the same name as the method that points to the method. The code is the same, but now instead of a codeblock, the method is a function in the class. Methods have the advantage of being easier to maintain and subclass.

Programs

Topic group Related topics

A program contains any combination of the following items:

- Statements to be executed
- Functions and classes that may be called
- Comments

The *dBASE Plus* compiler also supports a standard language preprocessor, so a program that is run by *dBASE Plus* may contain preprocessor directives. These directives are not part of the dBL language; instead they form a separate simple language that can affect the code compilation process, and are explained later.

Program files

Topic group Related topics

A program file may have any file-name extension, although there are a number of defaults:

A program containing a form is .WFM
A program containing a report is .REP
Any other program is .PRG

These file-name extensions are assumed by the Navigator and the Source Editor.

When a program is compiled into byte code by *dBASE Plus*, it stores the byte code in a file with the same name and extension, but it changes the last character of the extension to the letter "O": .PRG becomes .PRO, .WFM becomes .WFO, and .REP becomes .REO.

Program execution

Topic group Related topics

Use the DO command to run a program file, or double-click the file in the Navigator. If you run the program through the Navigator, the equivalent DO command will be streamed out to the Command window and executed. You can also call a .PRG program by name with the call operator, the parentheses, in the Command window; for example,

```
sales_report( )
```

will attempt to execute the file SALES_REPORTS.PRG. Since the operating system is not case-sensitive about file names when searching for files, neither is *dBASE Plus*.

A basic program simply contains a number of dBL statements, which are executed once in the order that they appear in the program file, from the top down. For example, the following four statements remember the current directory, switch to another directory, execute a report, and switch back to the previous directory:

```
cDir = set( "DIRECTORY" )
cd C:\SALES
do DAILY.REP
cd &cDir
```

Control statements, discussed earlier, are acted upon as they occur; they may affect the execution of the code that they contain. Some statements may be executed only when a certain condition is true and other statements may be executed more than once in a loop. But even within these control statements, the execution is still basically the same, from the top down.

When and if there are no more statement to execute, the program ends, and control returns to where the program was called. For example, if the program was executed from the Command window, then control returns to the Command window and you can do something else.

Functions and classes

Topic group Related topics

Functions and classes affect execution in two ways. First, when a function or class definition is encountered in the straight top-down execution of a program, execution in that program is terminated.

The second effect is that when a function, class constructor, or method is called, execution jumps into that function or class, executes that code in the usual top-down fashion, then goes back to where the call was made and continues where it left off.

Comments

Topic group Related topics

Use comments to include notes to yourself or others. The contents of a comment do not follow any dBL rules; include anything you want. Comments are stripped out at the beginning of the program compilation process.

A program will typically contain a group of comments at the beginning of the file, containing information like the name of the program, who wrote it and when, version information, and instructions for using it. But the most important use for comments is in the code itself, to explain the code—not obvious things like this:

```
n++ // Add one to the variable n
```

(unless you're writing example code to explain a language) but rather things like what you're doing in the overall scheme of the program, or why you decided to do something in a particular way. Decisions that are obvious to you when you write a statement will often completely bewilder you a few months later. Write comments so that they can be read by others, and put them in as you code, since there's rarely time to add them in after you're done, and you may have forgotten what you did by then anyway.

Preprocessor directives

Topic group Related topics

A preprocessor directive must be on its own line, and starts with the number sign (#).

Because preprocessor directives are not part of the dBL language, you cannot execute them in the Command window.

For more information about using preprocessor directives, see Preprocessor.

A simple program

Topic group Related topics

Here is a simple program that creates an instance of the RegPolygon class, changes the length of a side, and displays the perimeter:

```
// Polygon.prg
// A simple program example
//
local poly
poly = new RegPolygon( )
poly.length = 4
? poly.perimeter( ) // Displays 12

class RegPolygon
this.sides = 3 // Default number of sides
this.length = 1 // and default length
function perimeter( )
return this.sides * this.length

endclass
```

Language Syntax

Syntax conventions

Topic group Related topics

The *Language Reference* uses specific symbols and conventions in presenting the syntax of dBL language elements.

This section explains dBL syntax notation and provides an example of the various elements of the language syntax.

Syntax notation

Topic group Related topics

Statements, methods, and functions are described with syntax diagrams. These syntax diagrams consist of a least one fixed language element—the one being documented—and may include arguments, which are enclosed in angle brackets (< >).

The dBL language is not case-sensitive.

The following table describes the symbols used in syntax:

Symbol	Description
< >	Indicates an argument that you must supply
[ ]	Indicates an optional item
|	Indicates two or more mutually exclusive options
...	Indicates an item that may be repeated any number of times

Arguments are often expressions of a particular type. The description of an expression argument will indicate the type of argument expected, as listed in the following table:

Descriptor	Type
expC	A character expression
expN	A numeric expression
expL	A logical or boolean expression; that is, one that evaluates to *true* or *false*
expD	A date expression
exp	An expression of any type
oRef	An object reference

All the arguments and optional elements are described in the syntax description.

Unlike legacy dBASE command and function keywords, which are shown in uppercase letters, property names are capitalized differently. Property names are camel-capped, that is, they contain both uppercase and lowercase letters if the name consists of more than one word. If the property is a method, the name is followed by parentheses. Examples of properties include *onAppend*, *onRightMouseDown*, *checked*, and *close*( ).

These conventions help you differentiate the language elements; for example,

DELETE is a command
delete is a property
DELETED( ) is a function
delete( ) is a method

These typographical conventions are for readability only. When writing code, you can use any combination of uppercase and lowercase letters.

Note

In dBL, you must refer to classes and properties by their full names. However, you can still abbreviate some keywords in the dBL language to the first four characters, though for reasons of readability and clarity such abbreviation is not recommended.

Syntax example

Topic group Related topics

The syntax entries for the EXTERN statement illustrate all of the syntax symbols:

EXTERN [CDECL | PASCAL | STDCALL] <return type> <function name>
([<parameter type> [, <parameter type> ...]])
<filename>

The square brackets enclosing the calling convention, [CDECL | PASCAL | STDCALL], means the item is optional. The pipe character between the three calling conventions is an "or" indicator. In other words, if you want to use a calling convention, you must choose one of the three.

<return type> and <function name> are both required arguments.

The parentheses are fixed language elements, and thus also required. Inside the parentheses are optional <parameter type> arguments, as indicated by the square brackets.

The location of the comma inside the second square bracket indicates that the comma is needed only if more than one <parameter type> is specified.

The ellipsis (...) at the end means that any number of parameter type arguments may be specified (with a comma delimiter, if more than one is used).

<filename> is a required argument.

A simple EXTERN statement with neither of the two optional elements would look like this:

```
extern CINT angelsOnAPin( ) ANSWER.DLL
```

The <return type> argument is CINT, and the <function name> is angelsOnAPin.

A more complicated EXTERN statement with a calling convention and parameters would look like this:

```
extern PASCAL CLONG wordCount( CPTR, CLOGICAL ) ANSWER.DLL
```

Capitalization guidelines

Topic group Related topics

The following guidelines describe the standard capitalization of various language elements. Although dBL is not a case-sensitive language, you are encouraged to follow these guidelines in your own scripts.

Commands and built-in functions are shown in uppercase in descriptions so that they stand out, but are all lowercase in code examples.

Class names start with a capital letter. Multiple-word class names are joined together without any separators between the words, and each word starts with a capital letter. For example,

Form
PageTemplate

Property, event, and method names start with a lowercase letter. If they are multiple-word names, the words are joined together without any separators between the words, and each word (except the first) starts with a capital letter. They also appear italicized in the *Language Reference*. For example,

color
dataLink
showMemoEditor( )

Variable and function names are capitalized like property names.

Manifest constants created with the #define preprocessor directive are all uppercase, with underscores between words. For example,

```
ARRAY_DIR_NAME
NUM_REPS
```

Field names and table names from DBF tables are in all uppercase in code so that they stand out.

Operators and Symbols

Operators and symbols

Topic group Related topics

An operator is a symbol, set of symbols, or keyword that specifies an operation to be performed on data. Data is supplied in the form of arguments, or *operands*.

For example, in the expression "total = 0", the equal sign is the operator and "total" and "0" are the operands. In this expression, the numeric operator "=" takes two operands, which makes it a *binary* operator. Operators that require just one operand (such as the numeric increment operator "++") are known as *unary* operators.

Operators are categorized by type. dBL's operators are classified as follows:

Operator symbols	Operator category
`= := += -= *= /= %=`	Assignment
`= == <> # > < >= <= $`	Comparison
`+ -`	String concatenation
`+ - * / % ^ ** ++ --`	Numeric
`AND OR NOT`	Logical
`. [] NEW ::`	Object
`( )`	Call, Indirection
`->`	Alias
`&`	Macro

Most symbols you see in dBL code are operators, but not all. Quotation marks, for example, are used to denote literal strings and thus are part of the representation of a data type. Since they don't act upon data, they're a "non-operational" symbol.

You can use the following non-operational symbols in dBL code:

Symbols	Name/meaning
`;`	Statement separator, line continuation
`// &&`	End-of-line comment
`*`	Full-line comment
`/* */`	Block comment
`{} {;} {||}`	Literal date/literal array/codeblock markers
`"" '' []`	Literal strings
`::`	Name/database delimiters

#	Preprocessor directive

Finally, the following symbols are used as dBL commands when they are used to begin a statement:

Symbols	Name/meaning
? ??	Displays streaming output
!	Runs program or operating system command

Operator precedence

Topic group Related topics

dBL applies strict rules of precedence to compound expressions. In expressions that contain multiple operations, parenthetical groupings are evaluated first, with nested groupings evaluated from the "innermost" grouping outward. After all parenthetical groupings are evaluated, the rest of the expression is evaluated according to the following operator precedence:

Order of precedence (highest to lowest)	Operator description or category
&	Macro
(*expression*)	Parenthetical grouping, all expressions
->	Alias
( ) [] . NEW ::	Object operators: call; member (square bracket or dot); *new*; scope resolution
+ – ++ – –	Unary plus/minus, increment/decrement
^ **	Exponentiation
* / %	Multiply, divide, modulus
+ –	Addition, subtraction
= == <> # < <= > >= $	Comparison
NOT	Logical Not
AND	Logical And
OR	Logical Or
= := += –= *= /= %=	Assignment

In compound expressions that contain operators from the same precedence level, evaluation is conducted on a literal left-to-right basis. For example, no operator precedence is applied in the expressions 21/7*3 and 3*21/7 (both return 9).

Here's another example:

```
4+5*(6+2*(8-4)-9)%19>=11
```

This example is evaluated in the following order:

```
8–4=4
2*4=8
6+8=14
14-9=5
5*5=25
25%19=6
```

```
4+6=10
```

The result is the logical value *false*.

Assignment operators

Topic group Related topics

Assign/create operator: **=**

Assignment-only operator: **:=**

Arithmetic assignment operators: += –= *= /= %=

Syntax

x = n

y = x

x += y

Description

Assignment operators are binary operators that assign the value of the operand on the right to the operand on the left.

The standard assignment operator is the equal sign. For example, *x* = 4 assigns the value 4 to the variable *x*, and *y*= *x* assigns the value of the variable *x* (which must already have an assigned value) to the variable *y*. If the variable or property on the left of the equal sign does not exist, it is created.

To prevent the creation of a variable or property if it does not exist, use the assignment-only **:=** operator. This operator is particularly useful when assigning values to properties. If you inadvertently misspell the name of the property with the = operator, a new property is created; your code will run without error, but it will not behave as you intended. By using the **:=** operator, if the property (or variable) does not exist, an error occurs.

The arithmetic assignment operators are shortcuts to self-updating arithmetic operations. For example, the expression *x* += *y* means that *x* is assigned its own value plus that of *y*(*x* = *x* + *y*).Both operands must already have assigned values, or an error occurs. Thus, if the operand *x* has already been assigned the value 4 and *y* has been assigned the value *6*, the expression *x* += *y* returns 10.

+ operator

Topic group Related topics Example

Addition, concatenation, unary positive operator.

Syntax

n + m

date + n

"str1" + "str2"

"str" + x

x+ "str"

+n

Description

The "plus" operator performs a variety of additive operations:

It adds two numeric values together.

You may add a number to a date (or vice-versa). The result is the day that many days in the future (or the past if the number is negative). Adding any number to a blank date always results in a blank date.

It concatenates two strings.

You may concatenate any other data type to a string (or vice versa). The other data type is converted into its display representation:

Numbers become strings with no leading spaces. Integer values eight digits or less have no decimal point or decimal portion. Integer values larger than eight digits and non-integer values have as many decimals places as indicated by SET DECIMALS.

The logical values *true* and *false* become the strings "true" and "false".

Dates (primitive dates and Date objects) are converted using DTOC( ).

Object references to arrays are converted to the word "Array".

References to objects of all other classes are converted to the word "Object".

Function pointers take on the form "Function: " followed by the function name.

Note

Adding the value *null* to anything (or anything to *null*) results in the value *null*.

The plus sign may also be used as a unary operator to indicate no change in sign, as opposed to the unary minus operator, which changes sign. Of course, it is generally superfluous to indicate no change in sign; the unary plus is rarely used.

- operator

Topic group Related topics Example

Subtraction, concatenation, unary negative operator.

Syntax

n - m

date - n

date - date

"str1" - "str2"

"str" - x

x - "str"

-n

Description

The "minus" operator is similar to the "plus" operator. It subtracts two numbers, and subtracts days from a date. You may also subtract one date from another date; the result is the number of days between the two dates. If you subtract a blank date from another date, the result is always zero.

The minus symbol is also used as the unary negation operator, to change the sign of a numeric value.

You may concatenate two strings, or a string with any other data type, just like with the plus operator. The difference is that with the minus operator, the trailing blanks from the first operand are removed before the concatenation, and placed at the end of the result. This means that the concatenation with either the plus or minus results in a string with the same length, but with the minus operator, the trailing blanks are combined at the end of the result.

If you want to trim field values when creating an expression index for a DBF table, use the minus operator.

Numeric operators

Topic group Related topics

Binary numeric operators: + – * / % ^ **

Unary numeric operators: ++ – –

Syntax

n + m

n++

n – –

++n

n - m

n * m

n / m

n % m

n ^ m

n ** m

--n

Description

Perform standard arithmetic operations on two operands, or increment or decrement a single operand.

All of these operators take numeric values as operands. The + (plus) and - (minus) symbols can also be used to concatenate strings.

As binary numeric operators, the +, –, *, and / symbols perform the standard arithmetic operations addition, subtraction, multiplication and division.

The modulus operator returns the remainder of an integral division operation on its two operands. For example, 50%8 returns 2, which is the remainder after dividing 50 by 8.

You may use either ^ or ** for exponentiation. For example, 2^5 is 32.

The increment/decrement operators ++ and – – take a variable or property and increase or decrease its value by one. The operator may be used before the variable or property as a prefix operator, or afterward as postfix operator. For example,

```
n = 5 // Start with 5
? n++ // Get value (5), then increment
? n // Now 6
? ++n // Increment first, then get value (7)
? n // Still 7
```

If the value is not used immediately, it doesn't matter whether the ++/– – operator is prefix or postfix, but the convention is postfix.

Logical operators

Topic group Related topics

Binary logical operators: AND OR

Unary logical operator: NOT

Syntax

a AND b

a OR b

NOT b

Description

The AND and OR logical operators return a logical value (*true* or *false*) based on the result of a comparison of two operands. In a logical AND, both expressions must be *true* for the result to be *true*. In a logical OR, if either expression is *true*, or both are *true*, the result is *true*; if both expressions are *false*, the result is *false*.

When *dBASE Plus* evaluates an expression involving AND or OR, it uses short-circuit evaluation:

false AND <any expL> is always *false*
true OR <any expL> is always *true*

Because the result of the comparison is already known, there is no need to evaluate <any expL>. If <any expL> contains a function or method call, it is not called; therefore any side effects of calling that function or method do not occur.

The unary NOT operator returns the opposite of its operand expression. If the expression evaluates to *true*, then NOT *exp* returns *false*. If the expression evaluates to *false*, NOT *exp* returns *true*.

You may enclose the logical operators in dots, that is: .AND., .OR., and .NOT. The dots are required in earlier versions of dBASE.

Comparison operators

Topic group Related topics Example

Comparison operators compare two expressions. The comparison returns a logical *true* or *false* value. Comparing logical expressions is allowed, but redundant; use logical operators instead.

dBASE Plus automatically converts data types in a comparison, using the following rules:

1. If the two operands are the same type, they are compared as-is.
   - If either operand is a numeric expression, the other operand is converted to a number:

If a string contains a number only (leading spaces are OK), that number is used, otherwise it is interpreted as an invalid number.

The logical value *true* becomes one; *false* becomes zero.

All other data types are invalid numbers.

All comparisons between a number and an invalid number result in *false*.

- If either operand is a string, the other operand is converted to its display representation:

Numbers become strings with no leading spaces. Integer values eight digits or less have no decimal point or decimal portion. Integer values larger than eight digits and non-integer values have as many decimals places as indicated by SET DECIMALS.

The logical values *true* and *false* become the strings "true" and "false".

Dates (primitive dates and Date objects) are converted using DTOC( ).

Object references to arrays are converted to the word "Array".

References to objects of all other classes are converted to the word "Object".

Function pointers take on the form "Function: " followed by the function name.

- All other comparisons between mismatched data types return *false*.

These are the comparison operators:

Operator	Description
==	Exactly equal to
=	Equal to or Begins with
<> or #	Not equal to or Doesn't begin with
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
$	Contained in

When comparing dates, a blank date comes after (is greater than) a non-blank date.

When comparing Date objects, the date/time they represent are compared; they may be earlier, later, or exactly the same. For all other objects, only the equality tests makes sense. It tests whether two object references refer to the same object.

String equality comparisons are case-sensitive and follow the rules of SET EXACT. The == operator always compares two strings as if SET EXACT is ON. The other equality operators (=, <>, #) use the current setting of SET EXACT. When SET EXACT is ON, trailing blanks in either string are ignored in the comparison. When SET EXACT is OFF (the default), the = operator act like a "begins with" operator: the string on the left must begin with the string on the right. The <> and # operators act like "does not begin with" operators. Note that there is no single genuinely exactly equal comparison for strings in *dBASE Plus*.

It is recommended that you leave SET EXACT OFF so that you have the flexibility of doing an "exact" comparison or a "begins with" comparison as needed. By definition, all strings "begin with" an empty string, so when checking if a string is empty, always put the empty string on the left of the equality operator.

Warning!

For compatibility with earlier versions of dBASE, if the string on the right of the = operator is (or begins with) CHR(0) and SET EXACT is OFF, then the comparison always returns *true*. When checking for CHR(0), always use the == operator.

The $ operator determines if one string is contained in, or is a substring of, another string. By definition, an empty string is not contained in another string.

Object operators

Topic group Related topics

Object operators are used to create and reference objects, properties, and methods. Here are the Object operators:

Operator	Description
NEW	Creates a new instance of an object
[ ]	Index operator, which accesses the contents of an object through a numeric or string value

. (period)	Dot operator, which accesses the contents of an object through an identifier name
::	Scope resolution operator, to reference a method in a class or call a method from a class.

NEW operator

The NEW operator creates an object or instance of a specified class.

The following is the syntax for the NEW operator:

```
[<object reference> =] new <class name>([<parameters>])
```

The <object reference> is a variable or property in which you want to store a reference to the newly created object.

Note that the reference is optional syntactically; you may create an object without storing a reference to it. For most classes, this results in the object being destroyed after the statement that created it is finished, since there are no references to it.

The following example shows how to use the NEW operator to create a Form object from the Form class. A reference to the object is assigned to the variable customerForm:

```
customerForm = new Form( )
```

This example creates and immediately uses a Date object. The object is discarded after the statement is complete:

```
? new Date( ).toGMTString( )
```

Index operator

The index operator, [ ], accesses an object's properties or methods through a value, which is either a number or a character string. The following shows the syntax for using the index operator (often called the array index operator):

```
<object reference>[<exp>]
```

You typically use the index operator to reference elements of array objects, as shown in the following example:

```
aScores = new Array(20) // Create a new array object with 20 elements
aScores[1] = 10 // Change the value of the 1st element to 10
? aScores[1] // Displays 10 in results pane of Command window
```

Dot operator

The dot operator, ("."), accesses an object's properties, events, or methods through a name. The following shows the syntax for using the dot operator:

```
<object reference>[.<object reference> ...].<property name>
```

Objects may be nested: the property of an object may contain a reference to another object, and so on. Therefore, a single property reference may include many dots.

The following statements demonstrate how you use the dot operator to assign values:

```
custForm = new Form( ) // Create a new form object
custForm.title = "Customers" // Set the title property of custForm
custForm.height = 14 // Set the height property of custForm
```

If an object contains another object, you can access the child object's properties by building a path of object references leading to the property, as the following statements illustrate:

```
custForm.addButton = new Button(custForm) // Create a button in the custForm form
custForm.addButton.text = "Add" // Set the text property of addButton
```

Scope resolution operator

The scope resolution operator (::, two colons, no space between them) lets you reference methods directly from a class or call a method from a class.

The scope resolution operator uses the following syntax:

```
<class name>|class|super::<method name>
```

The operator must be preceded by either an explicit class name, the keyword CLASS or the keyword SUPER. CLASS and SUPER may be used only inside a class definition. CLASS refers to the class being defined and SUPER refers to the base class of the current class, if any.

<method name> is the method to be referenced or called.

Scope resolution searches for the named method, starting at the specified class and back through the class's ancestry. Because SUPER starts searching in a class's base class, it is used primarily when overriding methods.

Call, indirection, grouping operator

Topic group Related topics

Parentheses are used to call functions and methods, and to execute codeblocks. For example:

```
MyClass::MyMethod
```

is a function pointer to a method in the class, while

```
MyClass::MyMethod( )
```

actually calls that method. Any parameters to include in the call are placed inside the parentheses. Multiple parameters are separated by commas. Here is an example using a codeblock:

```
rootn = {|x,n| x^(1/n)} // Create expression codeblock with two parameters
? rootn( 27, 3 ) // Displays cube root of 27: 3
```

Some commands expect the names of files, indexes, aliases, and so forth to specified directly in command—"bare"—not in a character expression. Therefore, you cannot use a variable directly. For example, the ERASE command erases a file from disk. The following code will not work:

```
cFile = getfile( "*.*", "Erase file" ) // Store filename to variable
erase cFile // Tries to erase file named "cFile"
```

because the ERASE command tries to erase the file with the name of the variable, not the contents of the variable. To use the variable name in the file, enclose the variable in parentheses. In these commands, the parentheses evaluate the indirect file reference, and when used in this way, they are referred to as indirection operators:

```
erase ( cFile ) // Spaces inside parentheses optional
```

Macro substitution also works in these cases, but macro substitution can be ambiguous. Indirection operators are recommended in commands where they are allowed.

Finally, parentheses are also used for grouping in expressions to override or emphasize operator precedence. Emphasizing precedence simply means making the code more readable by explicitly grouping expressions in the normal order they are evaluated, so that you don't need to remember all the precedence rules to understand an expression. Overriding precedence uses the parentheses to change the order of evaluation. For example:

```
? 3 + 4 * 5 // Multiplication first, result is 23
? ( 3 + 4 ) * 5 // Do addition first, result is 35
```

Alias operator

Topic group Related topics Example

Designates a field name in a specific work area, or a private or public variable.

Syntax

alias->name

Description

When using a name that may be a variable or the name of a field in the current work area, the name is matched in the following order:

1. Local or static variable
    - Field name
    - Private or public variable

To resolve the ambiguity, or to refer to a field in another work area, use the alias operator. Aliases are not case-sensitive.

Private and public variables are referenced by the alias M. Use the alias of the specific work area to identify a particular field. Local and static variables cannot use the alias operator; you must use the variable alone.

Macro operator

Topic group Related topics Example

Substitutes the contents of a private or public string variable during the evaluation of a statement.

Syntax

&<character variable>[.]

Description

Macro substitution with the & operator allows you to change the actual text of a program statement at runtime. This capabilities allows you to overcome certain syntactic and architectural limitations in dBL.

The mechanics of macro substitution are as follows. When compiling a statement, in a program or for immediate execution in the Command window, *dBASE Plus* looks for any single & symbols in the statement. (Double ampersands [&&] denote end-of-line comments.) If something that looks like it could be a variable name—that is, a word made up of letters, numbers, and underscores—immediately follows the & symbol, its location is noted during compilation. If a period (.) happens to immediately follow the word, that period is considered to be a macro terminator.

When the statement is executed, *dBASE Plus* searches for a private or public variable with that name. If that variable exists, and that variable is a character variable, the contents of that variable are substituted in the statement in the place of the & symbol, the variable name, and the terminating period, if any. This is referred to as macro substitution. If no private or public variable with that name can be found, or if the variable is not a character variable, nothing happens; the statement is executed as-is.

Note

The & character is also used as the pick character in the *text* property of some form and menu components. For example, if you use the string "&Close" to designate the letter C as the pick character, if you happen to have a private or public variable named close, it will be substituted.

If macro substitution occurs, one of two things can happen:

Some commands expect certain kinds macro substitution. If the substitution is one of those cases, the command can immediately use the substituted value. For example, SET commands which expect either ON or OFF as the final word in the statement are optimized in this way.

If the substituted value is not an expected case, or if the command or statement does not expect macro substitution, the entire statement in its new form is recompiled on-the-fly and executed.

Recompiling the statement takes a small amount of time that is negligible unless you are constantly recompiling in a loop. Also, local and static variables may be out-of-scope when a recompiled statement is executed.

You cannot use the & operator immediately after a dot operator. You also cannot have the & and dot operators on the left side of an assignment operator; that is, you cannot assign to a property that is partially resolved with macro substitution. If you do either of these, a compile-time error occurs. You can assign to a property that is completely resolved with macro substitution, or use the STORE command instead of an assignment operator.

The macro terminator (a period, the same character as the dot operator) is required if you want to abut the macro variable name with a letter, number, underscore or dot operator. Compare the following examples:

```
&ftext // The macro variable ftext
&f.text // The macro variable f followed by the word text
&f..text // The macro variable f followed by the dot operator and the word text
```

Non-operational symbols

Topic group Related topics

Though they don't act upon data or hold values in themselves, non-operational symbols have their own purpose in dBL code and in the interpretation of programs.The symbols are:

Symbols	Name/meaning
`;`	Statement separator, line continuation
`// &&`	End-of-line comment
`*`	Full-line comment
`/* */`	Block comment
`{} {;} {||}`	Literal date/literal array/codeblock markers
`"" '' []`	Literal strings
`::`	Name/database delimiters
`#`	Preprocessor directive

String delimiters

Topic group Related topics

Enclose literal strings in either:

1. A set of single quote marks,
   - A set of double quote marks, or
   - A set of square brackets

The following example simply assigns the string "literal text" to the variable *xString*:

```
xString = "literal text"
```

To use a string delimiter in a literal string, use a different set of delimiters to delimit the string. For example:

```
? [There are three string delimiters: the ', the ",] + " and the []"
```

Note that the literal string had to be broken up into two separate strings, because all three kinds of delimiters were used.

Name/database delimiters

Topic group Related topics

If the name of a variable or a field in a work area contains a space, you may enclose the name in colons, for example:

```
local :a var:
:a var: = 4
? :a var: // Displays 4
```

Creating variables with spaces in them is strongly discouraged, but for some table types, it is not unlikely to get field names with spaces. If you create automem variables for that table, those variables will also have spaces.

However, if you're using the data objects instead of the Xbase DML, the fields are contained in a *fields* array and are referenced by name. The field name is a character expression, so you don't have to do anything different if the field name contains a space. The colons are not used.

You may also use colons when designating a table in a database. The name of the database is enclosed in colons before the name of the table, in the form:

```
:database:table
```

For example:

```
use :IBLOCAL:EMPLOYEE // IBLOCAL is sample Interbase database
```

Comment symbols

Topic group Related topics

Two forward slashes (//, no space between them) indicate that all text following the slashes (until the next carriage return) is a comment. Comments let you provide reference information and notes describing your code:

```
x = 4 * y // multiply the value of y by four and assign the result to variable x
```

Two ampersands (&&) can also be used for an end-of-line comment, but they are usually seen in older code.

If an asterisk (*) is the first character in a statement, the entire line is considered a comment.

A pair of single forward slashes with "inside" asterisks (/* */) encloses a block comment that can be used for a multi-line comment block:

```
/* this is the first line of a comment block
this is more of the comment
this is the last line of the comment block */
```

You can also use the pair for a comment in the middle of a statement:

```
x = 1000000 /* a million! */ * y
```

Comment blocks cannot be nested. This example shows improper usage:

```
/* this is the first line of a comment block
this is more of the same /* this nested comment will cause problems*/
this is the last line of the comment block */
```

After the opening block marker, *dBASE Plus* ends the comment at the next closing block marker it finds, which means that only the section of the comment from "this is the first line" to the word "problems" will be interpreted as a comment. The unenclosed remainder of the block will generate an error.

Statement separator, line continuation

Topic group Related topics

There is normally one statement per line in a program file. Use the semicolon to either:

- Combine multiple statements on a single line, or
- Create a multi-line statement

For example, a DO...UNTIL loop usually takes more than two lines: one for the DO, one for the UNTIL condition, and one or more lines in the loop body. But suppose all you want to do is loop until the condition is *true*; you can combine them using the semicolon as the statement separator:

```
do ; until rlock( ) // Wait for record lock
```

Long statements are easier to read if you break them up into multiple lines. Use the semicolon as the last non-comment character on the line to indicate that the statement continues on the next line. When the program is compiled, the comments are stripped; then any line that ends with a semicolon is tacked onto the beginning of the next line. For example, the program:

```
? "abc" + ; // A comment
"def" + ;
ghi
```

is compiled as

```
? "abc" + "def" + ghi
```

on line 3 of the program file. Note that the spaces before the semicolons and the spaces used to indent the code are not stripped. If an error occurs because there is no variable named ghi, the error will be reported on line 3.

Codeblock, literal date, literal array symbol

Topic group Related topics

Braces ({ }) enclose codeblocks, literal dates, and literal array elements. They must always be paired. The following examples show how braces may be used in dBL code.

Literal dates are interpreted according to the current settings of SET DATE and SET EPOCH:

```
dMoon = {07/20/69} // July 20, 1969 if SET DATE is MDY and SET EPOCH is 1950
```

To enclose arrays

```
a = {1,2,3}
? a[2] // displays 2
```

To assign a statement codeblock to an object's event handling property

```
form.onOpen = {;msgbox("Warning: You are about to enter a restricted area.")}
```

To assign an expression codeblock to a variable, and pass parameters to it

```
c = {|x| x*9}
```

```
? c(4) // returns 36
// or
q = {|n| {"1st","2nd","3rd"}[n]}
? q(2) // displays "2nd"
```

To assign an expression codeblock to a variable, without passing parameters

```
c = {|| 4*9} // pipes (||) must be included in an expression codeblock,
// even if a parameter is not being passed
? c( ) // returns 36
```

Preprocessor directive symbol

Topic group Related topics

The number sign (#) marks preprocessor directives, which provide instructions to the *dBASE Plus* compiler. Preprocessor directives may be used in programs only.

Use directives in your dBL code to perform such compile-time actions as replacing text throughout your program, perform conditional compilations, include other source files, or specify compiler options.

The symbol must be the non-blank first character on a line, followed by the directive (with no space), followed by any conditions or parameters for the directive.

For example, you might use this statement:

```
#include "IDENT.H"
```

to include a source file named IDENT.H (the "H" extension us generally used to identify the file as a "header" file) in the compilation. The included file might contain its own directives, such as constant definitions:

```
//file IDENT.H: constant definitions for MYPROG
#define COMPANY_NAME "Nobody's Business"
#define NUM_EMPLOYEES 1
#define COUNTRY "Liechtenstein"
```

For a complete listing of all *dBASE Plus* preprocessor directives, along with syntax and examples for each, see Preprocessor.

IDE Language Elements

IDE overview

Topic group

This section of the *Language Reference* describes language elements that you use within the *dBASE Plus* integrated development environment (IDE) to programmatically create, modify, compile and build applications.

BUILD

Topic group Related topics

Creates a Windows executable file (.EXE) from your *dBASE Plus* object files and resources.

Syntax

BUILD FROM <*project or response file name*>

or

BUILD <filename>[, <filename> ...] /FROM <resp-filename>
[ICON <filename>] [SPLASH <filename>] [TO <exe-filename>]
[WEB] [INI [ON | OFF | ROAM]]
[UAC]

FROM *<project or response file name>*

Name of a *dBASE Plus* project or response file that contains the names of all object files and resources that are to be linked into your executable. If no extension is provided, .PRJ is assumed.

<filename list>

List of compiled program elements, separated by commas. If you provide a filename without an extension, .PRO (compiled program) is assumed.

ICON *<icon filename>*

Optional icon (.ICO) file used to identify your program in the Windows environment (e.g., when minimized or listed in the Windows Explorer or a program group).

SPLASH <bmp format filename>

Optional bitmap (.BMP) file that displays while your program loads.

TO *<executable filename>*

The name of the Windows executable file (.EXE) to create. If not specified, the base file name of the named project or response file (or the first file name in *<filename list>*) is used.

WEB

Specifies that an application will be used as a web application, rather than a desktop application.

When run, an application built using the WEB keyword will take advantage of optimizations built into PLUSrun.exe which allow it to load faster and use fewer resources than a non-WEB application. Please note that these optimizations restrict a web application from containing code to create, or use, visual components such as forms, buttons, toolbars, status bars, and other form components. Only non-visual objects such as sessions, data modules, queries, rowsets, non-visual objects and custom classes should be used.

In addition, when a web application .exe is run directly, rather than as a parameter to PLUSrun.exe, using the WEB parameter allows it to detect when it's been prematurely terminated by a Web server (as happens when an application takes too long to respond). If a premature termination occurs, PLUSrun.exe also terminates to prevent it from becoming stranded in memory.

To determine if an application was built using the WEB parameter, see the _app object's *web* property. For additional information, see "Startup optimizations for Web applications" and "Change to command line for PLUSrun.exe".

INI

INI or INI ON - indicates that the application will create and use an ini file. This is the same as NOT specifying an INI clause at all.

If _app.useUACPaths is True,

- the .ini file will be located in the path contained in _app.currentUserPath.

If _app.useUACPaths is False,

- the .ini file will be located in the path contained in _app.exeName

INI ROAM - indicates that the application will create and use an ini file under the path in _app.roamingUsersPath instead of the _app.currentUserPath

Note that the location of an .ini file can be overridden via the -c command line switch which can be used to specify an alternate folder in which to locate the .ini file.

UAC

When UAC is specified the resulting .exe is built with an embedded default to set _app.UseUACPaths to True when the .exe is run.

This embedded UAC setting overrides the runtime engine default set via the registry key: HKLM\SOFTWARE\dBASE\Plus\Series1\useUACPaths

However, the embedded setting can be overridden by:

1- setting a RuntimeApp specific registry setting in registry key:
HKLM\SOFTWARE\dBASE\Plus\RuntimeApps\<app file name>\useUACPaths
useUACPaths is a string value set to "Y" or "y" for 'True' and set to "N" or "n" for 'False'
(NOTE: the <app file name> is case sensitive so 'MyApp.exe' is NOT the same as 'myApp.exe')

Or

2 - by using the -v command line switch:
-v1 sets UseUACPaths to true
-v0 sets UseUACPaths to false

Description

Use the BUILD command to link compiled *dBASE Plus* program elements and supporting resources (such as bitmaps and icons) into a Windows executable (.EXE) file.

Though the new project file format is the default for build specifications, support for response (.RSP) files is offered for backward compatibility.

For Web based applications, it's important to use the WEB parameter. When a server terminates an application, the WEB parameter enables *dBASE Plus* to simultaneously terminate it's runtime.

Code Signing

dBASE has been upgraded so it can build .exe's that can be code signed.

New executables built with dBASE Plus will contain some additional information that will allow them to be loaded successfully with the newruntime engine whether or not they are signed with a digital signature.

The new dBASE runtime will check a dBASE built .exe for the new data. If found, it will be loaded using the digital signature safe way. If not found, it will be loaded the old way which will not support digital signatures.

The new runtime is therefore, backward compatible with executables built with prior versions of dBASE Plus.

In addition, executables built with the new version of dBASE Plus will work with older dBASE Plus runtime engines unless it requires features available only in the newer runtime engine.

CLEAR ALL

Topic group Related topics

Releases all user-defined memory variables and closes all open files.

Syntax

CLEAR ALL

Description

CLEAR ALL combines the CLEAR MEMORY and CLOSE ALL commands, releasing all user-defined memory variables, closing all open tables in the current workset, and all other files. For more information, see CLEAR MEMORY and CLOSE ALL.

Note

CLEAR ALL does not explicitly release objects. However, if the only reference to an object is in a variable, releasing the variable with CLEAR ALL in turn releases the object.

Use CLEAR ALL during development to clear all variables (and any objects that rely on those references) and close all files to reset your working environment. Because of the event-driven nature of *dBASE Plus*, CLEAR ALL is generally not used in programs.

CLOSE ALL

Topic group Related topics

Closes (almost) all open files.

Syntax

CLOSE ALL [PERSISTENT]

PERSISTENT

In addition to files closed by CLOSE ALL, the PERSISTENT designation closes files tagged PERSISTENT. Without the PERSISTENT designation, these files would not be affected.

Description

CLOSE ALL closes almost all open files, including:

- All databases opened by the Navigator and with OPEN DATABASE
- All tables opened (with USE) in all work areas in the current workset
- All files opened with low-level file functions, or a File object
- All procedure and library files opened with SET PROCEDURE and SET LIBRARY
- Any text streaming file opened by SET ALTERNATE

It does not close:

- The printer file specified by the SET PRINTER TO command
- Tables or databases opened through the data objects

Use CLOSE ALL during development to close files and reset your working environment without affecting any variables. To close all files and release all variables, use CLEAR ALL. Because of the event-driven nature of *dBASE Plus*, CLOSE ALL is generally not used in programs.

CLOSE FORMS

Topic group Related topics

Closes all open forms.

Syntax

CLOSE FORMS [<*form name list*>]

<form name list>

List of forms (wfm.) to close.

Description

Closes the specified forms when using <*form name list*>. Closes all forms when no list is specified. Executes the standard close routines for the forms and the objects that are contained in them.

COMPILE

Topic group Related topics

Compiles program files (.PRG, .WFM), creating object code files (.PRO, .WFO).

Syntax

COMPILE <filename 1> | <filename skeleton>
[AUTO]
[LOG <filename 2>]

<filename 1> | <filename skeleton>

The file(s) to compile. If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory only. If you specify a file without including its extension, *dBASE Plus* assumes .PRG.

AUTO

The optional AUTO clause causes the compiler to detect automatically which files are called by your program, and to recompile those files.

LOG <filename 2>

Logs the files that were compiled, and any compiler errors or warning messages to <filename 2>. The default extension for the log file is .TXT.

Description

Use COMPILE to explicitly compile or recompile program files without loading or executing them. *dBASE Plus* automatically compiles program source files into object (bytecode) files when they are loaded (with SET PROCEDURE or SET LIBRARY) or executed (with DO or the call operator). The compiled object files are created in the same directory as the source code files.

The file is compiled with coverage information if SET COVERAGE is ON or the file contains the

```
#pragma coverage(on)
```

directive.

When you compile a program, *dBASE Plus* detects any syntax errors in the source file and either logs the error in the LOG file, or displays an error message corresponding to the error in a dialog box that contains three buttons:

Cancel cancels compilation (equivalent to pressing Esc).

Ignore cancels compilation of the program containing the syntax error but continues compilation of the rest of the files that match <filename skeleton> if you specified a skeleton.

Fix lets you fix the error by opening the source code in an editing window, positioning the insertion point at the point where the error occurred.

CONVERT

Topic group Related topics Example

Adds a _dbaselock field to a table for storing multiuser lock information.

Syntax

CONVERT [TO <expN>]

TO <expN>

Specifies the length of the multiuser information field to add to the current table. The <expN> argument can be a number from 8 to 24, inclusive. The default is 16.

Description

Use CONVERT to add a special _dbaselock field to the structure of the current table. In general, CONVERT is a one-time operation required for each table that is shared in a multi-user environment.

Use the option TO <expN> to specify the length of the field. If you issue CONVERT without the TO <expN> option, the width of the field is 16. If you want to change the length of the _dbaselock field after using CONVERT, you can issue CONVERT again on the same table. To view the contents of the _dbaselock field, use LKSYS( ).

Note

You must use the table exclusively (USE...EXCLUSIVE) before issuing CONVERT. Any records marked as deleted with be lost during the CONVERT.

The _dbaselock field contains the following values:

Count	A 2-byte hexadecimal number used by CHANGE( )
Time	A 3-byte hexadecimal number that records the time a lock was placed
Date	A 3-byte hexadecimal number that records the date a lock was placed
Name	A 0- to 16-character representation of the login name of the user who placed a lock, if a lock is active

The count, time, and date portions of the _dbaselock field always make up its first 8 characters. If you accept the default 16-character width of the _dbaselock field, the login name is truncated to 8 characters. If you set the field width to fewer than 16 characters, the login name is truncated the necessary amount. If you set the width of <expN> to 8 characters, the login name doesn't appear at all.

Every time a record is updated, *dBASE Plus* rewrites the count portion of _dbaselock. If you issue CHANGE( ), *dBASE Plus* reads the count portion from disk and compares it to the previous value it stored in memory when the record was initially read. If the values are different, another user has changed the record, and CHANGE( ) returns *true*. For more information, see CHANGE( ).

LKSYS( ) returns the login name, date, and time portions of the _dbaselock field. If you place a file lock on the table containing the _dbaselock field, the value in the _dbaselock field of the first

record contains the information used by CHANGE( ) and LKSYS( ). For more information, see LKSYS( ).

Note

CONVERT doesn't affect SQL databases or Paradox tables.

CREATE

Topic group Related topics

Opens the Table designer to create or modify a table interactively.

Syntax

CREATE
[<filename> | ? | <filename skeleton>
[[TYPE] FOXPRO | PARADOX | DBASE]
[WIZARD | EXPERT [PROMPT]]

<filename> | ? | <filename skeleton>

The name of the table you want to create. Both CREATE ? and CREATE <filename skeleton> display a dialog box in which you can specify the name of a new table. The <filename> follows the standard Xbase DML table naming conventions.

If you don't specify a name, the table remains untitled until you save the file. If you specify an existing table name, *dBASE Plus* asks whether you want to overwrite it. If you reply no, nothing further happens.

[TYPE] FOXPRO | PARADOX | DBASE

Overrides the default table type set by SET DBTYPE. The TYPE keyword is included for readability only; it has no effect on the operation of the command.

PARADOX creates a Paradox table with a .DB extension.

FOXPRO creates a FoxPro table with a .DBF extension.

DBASE creates a DBF table with a .DBF extension.

```
CREATE MYTABLE PARADOX // Opens the table designer for "Mytable.db"
```

[WIZARD | EXPERT [PROMPT]]

If the PROMPT clause is used, a dialog appears asking if you want to use the Table designer or the Table wizard. You can then invoke either the designer or the wizard. The WIZARD clause without PROMPT causes the Table wizard to be invoked. You may use the keyword EXPERT instead of WIZARD.

```
CREATE MYTABLE PARADOX WIZARD // Opens the Table Wizard
CREATE MYTABLE PARADOX WIZARD PROMPT // Opens the New Table dialog
allowing a choice of using the Table Designer or the Table Wizard.
```

Description

CREATE opens the Table designer, an interactive environment in which you can create or modify the structure of a table, or the Table wizard, a tool that guides you through the process of creating tables. The type of table you create depends on the <filename> you specify, or the current database and the current setting of SET DBTYPE.

Create a table by defining the name, type, and size of each field. For more information on using the Table designer, see The Table designer window.

To modify an existing table, use the MODIFY STRUCTURE command.

CREATE COMMAND

Topic group Related topics

Displays a specified program file for editing, or displays an empty editing window.

Syntax

CREATE COMMAND [<filename> | ? | <filename skeleton>]

<filename> | ? | <filename skeleton>

The file to display and edit. The ? and <filename skeleton> options display a dialog box from which you can select a file. If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, *dBASE Plus* assumes .PRG. If you issue CREATE COMMAND without an option, *dBASE Plus* displays an untitled empty editing window.

Description

Use CREATE COMMAND to create new or edit existing program files. Use DO to execute program files.

If you're creating a new program file, CREATE COMMAND displays an empty editing window. If you specify an existing file, *dBASE Plus* asks whether you want to modify it. If you reply no, nothing further happens. Use the MODIFY COMMAND command to edit an existing file without being asked whether you want to modify it.

By default, CREATE COMMAND launches the *dBASE Plus* Source Editor. You can specify an alternate editor by using the SET EDITOR command or by changing the EDITOR setting in PLUS.ini. To do so, either use the SET command to specify the setting interactively, or enter the EDITOR parameter directly in PLUS.ini.

Note

dBASE Plus compiles programs before running them, and assigns the compiled files the same name as the original, but with the letter "O" as the last letter in the filename extension. For example, the compiled version of SALESRPT.PRG would be SALESRPT.PRO. If SALESPRT.PRO already exists, it is overwritten. For this reason, avoid using filename extensions ending in "O" in directories containing compiled programs.

CREATE DATAMODULE

Topic group

Opens the Data Module designer.

Syntax

CREATE DATAMODULE
[<filename> | ? | <filename skeleton>]
[CUSTOM] | [WIZARD | EXPERT [PROMPT]]

<filename> | ? | <filename skeleton>

The file to display and edit. The default extension is .DMD. The ? and <filename skeleton> options display a dialog box from which you can select a file. If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory. If you issue CREATE DATAMODULE without an option, *dBASE Plus* creates an untitled empty data module.

CUSTOM

Invokes the Custom Data Module designer instead of the Data Module designer. The default extension is .CDM instead of .DMD.

WIZARD | EXPERT [PROMPT]

If the PROMPT clause is used, a dialog appears asking if you want to use the Data Module designer or the Data Module wizard. You can then invoke either the designer or the wizard. The WIZARD clause without PROMPT causes the Table wizard to be invoked. You may use the keyword EXPERT instead of WIZARD.

You cannot combine the CUSTOM and WIZARD options; there is no Custom Data Module wizard.

Description

Use CREATE DATAMODULE to open the Data Module designer and create new or edit existing data modules. The Data Module designer automatically generates dBL program code that defines the data in the data module, and stores this code in an editable source code file with a .DMD extension. Use a dataModRef object to use a data module.

If you're creating a new data module, CREATE DATAMODULE displays an empty design surface. If you specify an existing file, *dBASE Plus* asks whether you want to modify it. If you reply no, nothing further happens. Use the MODIFY DATAMODULE command to edit an existing file without being asked whether you want to modify it.

CREATE FILE

Topic group Related topics

Displays a specified text file for editing, or displays an empty editing window.

Syntax

CREATE FILE [<filename> | ? | <filename skeleton>]

Description

CREATE FILE is identical to CREATE COMMAND, except that it defaults to displaying and editing text files, which have a .TXT extension (instead of program files, which have a .PRG extension).

CREATE FORM

Topic group Related topics

Opens the Form designer to create or modify a form.

Syntax

CREATE FORM
[<filename> | ? | <filename skeleton>]
[CUSTOM] | [WIZARD | EXPERT [PROMPT]]

<filename> | ? | <filename skeleton>

The form to create or modify. The default extension is .WFM. The ? and <filename skeleton> options display a dialog box from which you can select a file. If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory. If you issue CREATE FORM without an option, *dBASE Plus* creates an untitled empty form.

CUSTOM

Invokes the Custom Form designer instead of the Form designer. The default extension is .CFM instead of .WFM.

WIZARD | EXPERT [PROMPT]

If the PROMPT clause is used, a dialog appears asking if you want to use the Form designer or the Form wizard. You can then invoke either the designer or the wizard. The WIZARD clause without PROMPT causes the Form wizard to be invoked. You may use the keyword EXPERT instead of WIZARD.

You cannot combine the CUSTOM and WIZARD options; there is no Custom Form wizard.

Description

Use CREATE FORM to open the Form designer or Form wizard and create or modify a form interactively. The Form designer automatically generates dBL program code that defines the contents and format of a form, and stores this code in an editable source code file with a .WFM extension. DO the .WFM file to run the form.

If you're creating a new form, CREATE FORM displays an empty design surface. If you specify an existing file, *dBASE Plus* asks whether you want to modify it. If you reply no, nothing further happens. Use the MODIFY FORM command to edit an existing file without being asked whether you want to modify it.

You may invoke the Custom Form designer by specifying the CUSTOM keyword. A custom form is stored in a .CFM file, and does not have the standard bootstrap code that instantiates and opens a form when the file is executed. It is intended to be used as a base class for other forms. A single .CFM file may contain more than one custom form class definition. If there is more than one form class in the .CFM file, *dBASE Plus* presents a list of classes to modify.

By default, the Form designer creates a class made up of the name of the file plus the word "Form". For example, when creating STUDENT.WFM, the form class is named StudentForm. The Custom Form designer uses the word "CForm" instead; for example, in SCHOOL.CFM, the form class is named SchoolCForm.

See Using the Form and Report designers (overview) for instructions on using the Form designer.

CREATE LABEL

Topic group Related topics

Opens the Label designer to create or modify a label file.

Syntax

CREATE LABEL
[<filename> | ? | <filename skeleton>]
[WIZARD | EXPERT [PROMPT]]

<filename> | ? | <filename skeleton>

The label file to create or modify. The default extension is .LAB. The ? and <filename skeleton> options display a dialog box from which you can select a file. If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory. If you issue CREATE LABEL without an option, *dBASE Plus* creates an untitled label file.

WIZARD | EXPERT [PROMPT]

If the PROMPT clause is used, a dialog appears asking if you want to use the Label designer or the Label wizard. You can then invoke either the designer or the wizard. The WIZARD clause

without PROMPT causes the Label wizard to be invoked. You may use the keyword EXPERT instead of WIZARD.

Description

Use CREATE LABEL to open the Label designer and create new or edit existing labels. The Label designer automatically generates dBL program code that defines the contents and format of the labels, and stores this code in an editable source code file with a .LAB extension. DO the .LAB file to print the labels.

If you're creating a new label file, CREATE LABEL displays an empty design surface. If you specify an existing file, *dBASE Plus* asks whether you want to modify it. If you reply no, nothing further happens. Use the MODIFY LABEL command to edit an existing file without being asked whether you want to modify it.

CREATE MENU

Topic group Related topics

Opens the Menu designer to create or modify a menu file.

Syntax

CREATE MENU [<filename> | ? | <filename skeleton>]

<filename> | ? | <filename skeleton>

The menu file to create or modify. The default extension is .MNU. The ? and <filename skeleton> options display a dialog box from which you can select a file. If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory. If you issue CREATE MENU without an option, *dBASE Plus* creates an untitled menu file.

Description

Use CREATE MENU to open the Menu designer and create new or edit existing menus. The Menu designer automatically generates dBL program code that defines the contents of a menu, and stores this code in an editable source code file with a .MNU extension. To use the menu, assign the .MNU file name as the *menuFile* property of a form, or

```
DO <.MNU file> WITH <form reference>
```

to assign the menu to the form. The Menu designer always creates a menu named "root", so that when assigned to a form, it is referenced as form.root.

If you're creating a new menu file, CREATE MENU displays an empty design surface. If you specify an existing file, *dBASE Plus* asks whether you want to modify it. If you reply no, nothing further happens. Use the MODIFY MENU command to edit an existing file without being asked whether you want to modify it.

CREATE POPUP

Topic group Related topics

Opens the Popup Menu designer to create or modify a popup menu file.

Syntax

CREATE POPUP [<*filename*> | ? | <*filename skeleton*>]

<filename> | ? | <filename skeleton>

The popup menu file to create or modify. The default extension is .POP. The ? and <filename skeleton> options display a dialog box from which you can select a file. If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory. If you issue CREATE POPUP without an option, *dBASE Plus* creates an untitled popup menu file.

Description

Use CREATE POPUP to open the Popup Menu designer and create new or edit existing popup menus. The Popup Menu designer automatically generates dBL program code that defines the contents of a popup menu, and stores this code in an editable source code file with a .POP extension. To assign the popup menu to a form, create the popup as a property of the form with:

```
DO <.POP file> WITH <form reference>, <property name>
```

then assign the popup object to the form's *popupMenu* property.

If you're creating a new popup menu file, CREATE POPUP displays an empty design surface. If you specify an existing file, *dBASE Plus* asks whether you want to modify it. If you reply no, nothing further happens. Use the MODIFY POPUP command to edit an existing file without being asked whether you want to modify it.

CREATE PROJECT

Topic group Related topics

Syntax

CREATE PROJECT

Description

CREATE PROJECT opens the Project Explorer, where you can design a new project.

Use MODIFY PROJECT to open existing project.

CREATE QUERY

Topic group Related topics

Opens a new or existing query in the SQL designer.

Syntax

CREATE QUERY [<filename> | ? | <*filename skeleton*>]

<filename> | ? | <filename skeleton>

The SQL query file to create or modify. The default extension is .SQL. The ? and <filename skeleton> options display a dialog box from which you can select a file. If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory. If you issue CREATE QUERY without an option, *dBASE Plus* creates an untitled SQL query file.

Description

Use CREATE QUERY to open the SQL designer and create new or edit existing SQL queries. The SQL designer automatically generates an SQL statement that defines the query, and stores this statement in an editable source code file with a .SQL extension. The .SQL file can be run directly from the Navigator or used as the *sql* property of a Query object.

If you're creating a new SQL query file, CREATE QUERY displays an empty design surface. If you specify an existing file, *dBASE Plus* asks whether you want to modify it. If you reply no,

nothing further happens. Use the MODIFY QUERY command to edit an existing file without being asked whether you want to modify it.

CREATE REPORT

Topic group Related topics

Opens the Report designer to create or modify a report.

Syntax

CREATE REPORT
[<filename> | ? | <filename skeleton>]
[CUSTOM] | [WIZARD | EXPERT [PROMPT]]

<filename> | ? | <filename skeleton>

The report to create or modify. The default extension is .REP. The ? and <filename skeleton> options display a dialog box from which you can select a file. If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory. If you issue CREATE REPORT without an option, *dBASE Plus* creates an untitled empty report.

CUSTOM

Invokes the Custom Report designer instead of the Report designer. The default extension is .CRP instead of .REP.

WIZARD | EXPERT [PROMPT]

If the PROMPT clause is used, a dialog appears asking if you want to use the Report designer or the Report wizard. You can then invoke either the designer or the wizard. The WIZARD clause without PROMPT causes the Report wizard to be invoked. You may use the keyword EXPERT instead of WIZARD.

You cannot combine the CUSTOM and WIZARD options; there is no Custom Report wizard.

Description

Use CREATE Report to open the Report designer or Report wizard and create or modify a report interactively. The Report designer automatically generates dBL program code that defines the contents and format of a report, and stores this code in an editable source code file with a .REP extension. DO the .REP file to run the report.

If you're creating a new report, CREATE REPORT displays an empty design surface. If you specify an existing file, *dBASE Plus* asks whether you want to modify it. If you reply no, nothing further happens. Use the MODIFY REPORT command to edit an existing file without being asked whether you want to modify it.

You may invoke the Custom Report designer by specifying the CUSTOM keyword. A custom report is stored in a .CRP file, and does not have the standard bootstrap code that instantiates and renders a report when the file is executed. It is intended to be used as a base class for other reports. A single .CRP file may contain more than one custom report class definition. If there is more than one report class in the .CRP file, *dBASE Plus* presents a list of classes to modify.

See Using the Report and Report designers (overview) for instructions on using the Report designer.

DEBUG

Topic group Related topics

Opens the *dBASE Plus* Debugger.

Syntax

DEBUG
[<filename> | ? | <filename skeleton> [WITH <parameter list>]]

<filename> | ? | <filename skeleton>

The program file to debug. DEBUG ? and DEBUG <filename skeleton> display the Open Source File dialog box, from which you can select a file. If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, *dBASE Plus* assumes .PRG.

WITH <parameter list>

Specifies expressions to pass as parameters to a program. For information about parameter passing, see the description of PARAMETERS.

Description

Use DEBUG to open the Debugger and view or control program execution interactively. You must issue DEBUG in the Command window; the command has no effect in a program. If you issue DEBUG without any options, *dBASE Plus* opens the Debugger without loading a program file. (You can load a file to debug from the Debugger.)

To debug a function, open the program file that contains the function, and set a breakpoint at the FUNCTION or PROCEDURE line. When the function is called, the debugger will appear, at the breakpoint that you set.

If an unhandled exception or error occurs during program execution, the standard error dialog gives you the option of opening the Debugger at the line where the error occurred.

For more information, see Using the Debugger, which describes the Debugger in detail.

DISPLAY COVERAGE

Topic group Related topics

Displays the contents of a coverage file in the results pane of the Command window.

Syntax

DISPLAY COVERAGE <filename1> | ? | <filename skeleton 1>
[ALL]
[SUMMARY]
[TO FILE <filename2> | ? | <filename skeleton 2>]
[TO PRINTER]

<filename1> | ? | <filename skeleton 1>

The coverage file for the desired program. The ? and <filename skeleton 1> options display a dialog box from which you can select a coverage file. If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, *dBASE Plus* assumes .COV.

ALL

Includes the coverage files, if any, for all other program files that could be called by the main program file, adding to the display:

- The total number of logical blocks exercised in all the program files combined
- The percentage of logical blocks exercised in all the program files combined

SUMMARY

Excludes the logical blocks that were exercised. Without SUMMARY, both the logical blocks that were exercised, and the logical blocks not exercised are displayed. Use the SUMMARY option to find code that still needs to be exercised.

TO FILE <filename2> | ? | <filename skeleton 2>

Directs output to <filename2> in addition to the results pane of the Command window. By default, *dBASE Plus* assigns a .TXT extension to <filename2> and saves the file in the current directory. The ? and <filename skeleton 2> options display a dialog box in which you specify the name of the target file and the directory to save it in.

TO PRINTER

Directs output to the printer in addition to the results pane of the Command window.

Description

A coverage file contains the results of the coverage analysis of a program file. You cause *dBASE Plus* to analyze the execution of any code in a program file by compiling the program file with coverage, either by having SET COVERAGE ON when the program is compiled, or with the #pragma coverage(on) directive in the program file. A coverage file is created whenever any code in the program file is executed.

The coverage file has the same name as the program file, and changes the last letter of the extension to the letter "V"; unless the file is a .PRG, in which case the coverage file has an extension of .COV. For example, the coverage file for GRADES.PRG is GRADES.COV, and the coverage file for STUDENTS.WFM is STUDENTS.WFV.

The coverage file accumulates statistics whenever any code in the program file is executed. You will usually want to make sure that all logical blocks in your code have been exercised. You may erase the coverage file to restart the coverage analysis totals.

DISPLAY COVERAGE displays the results of the coverage analysis:

- Each logical block, and how many times it was exercised
- The total number of blocks, and the number of blocks that were tested
- The percentage of blocks tested

DISPLAY COVERAGE pauses when the results pane is full and displays a dialog box prompting you to display another screenful of information. Use the TO FILE clause to send the information to a file. Use the TO PRINTER clause to send the information to the printer. In either case, you can use SET CONSOLE OFF to suppress the display of the information in the results pane.

DISPLAY COVERAGE is the same as LIST COVERAGE, except that LIST COVERAGE does not pause with the first window of information but rather continuously lists the information until complete. This makes LIST COVERAGE more appropriate for outputting to a file or printer.

DISPLAY MEMORY

Topic group Related topics

Displays information about memory variables in the results pane of the Command window.

Syntax

DISPLAY MEMORY
[TO FILE <filename> | ? | <filename skeleton>]
[TO PRINTER]

TO FILE <filename> | ? | <filename skeleton>

Directs output to the text file <filename>, in addition to the results pane of the Command window. By default, *dBASE Plus* assigns a .TXT extension to <filename> and saves the file in the current directory. The ? and <filename skeleton> options display a dialog box in which you specify the name of the target file and the directory to save it in.

TO PRINTER

Directs output to the printer in addition to the results pane of the Command window.

Description

Use DISPLAY MEMORY to display the contents and size of a memory variable list. If you haven't used ON KEY or to reassign the F7 key, pressing F7 when the Command window has focus is a quick way to execute DISPLAY MEMORY.

DISPLAY MEMORY displays information about both user-defined and system memory variables. The following information on user-defined memory variables is displayed.

- Name
- Scope (public, private, local, static or hidden)
- Data type
- Value
- Number of active memory variables
- Number of memory variables still available for use
- Number of bytes of memory used by character variables
- Number of bytes of memory still available for user character variables
- Name of the program that initialized private memory variables

The following information on system memory variables is displayed.

- Name
- Scope (public, private, or hidden)
- Data type
- Current value

DISPLAY MEMORY pauses when the results pane is full and displays a dialog box prompting you to display another screenful of information. Use the TO FILE clause to send the information to a file. Use the TO PRINTER clause to send the information to the printer. In either case, you can use SET CONSOLE OFF to suppress the display of the information in the results pane.

DISPLAY MEMORY is the same as LIST MEMORY, except that LIST MEMORY does not pause with the first window of information but rather continuously lists the information until complete. This makes LIST MEMORY more appropriate for outputting to a file or printer.

DISPLAY STATUS

Topic group Related topics

Displays information about the current *dBASE Plus* environment in the results pane of the Command window.

Syntax

DISPLAY STATUS
[TO FILE <filename> | ? | <filename skeleton>]
[TO PRINTER]

TO FILE <filename> | ? | <filename skeleton>

Directs output to the text file <filename>, in addition to the results pane of the Command window. By default, *dBASE Plus* assigns a .TXT extension to <filename> and saves the file in

the current directory. The ? and <filename skeleton> options display a dialog box in which you specify the name of the target file and the directory to save it in.

TO PRINTER

Directs output to the printer in addition to the results pane of the Command window.

Description

Use DISPLAY STATUS to identify open tables and index files and to check the status of the SET commands. DISPLAY STATUS shows information related to the current session only.

If you haven't used ON KEY, SET, or SET FUNCTION to reassign the F6 key, pressing F6 when the Command window has focus is a quick way to execute DISPLAY STATUS.

DISPLAY STATUS displays the following information:

- Name and alias of open tables in each work area, and for each table:
  - Whether that table is the table in the currently selected work area
  - The language driver and character set of each open table
  - Names of all open indexes and their index key expressions in each work area
  - Master index, if any, in each work area
  - Locked records in each work area
  - Database relations in each work area
- Filter conditions in each work area
- The name of the SET LIBRARY file, if any
- The name of all open SET PROCEDURE files
- SET PATH file search path
- SET DEFAULT drive setting
- Current work area
- SET PRINTER setting
- Current language driver and character set
- DBTYPE setting
- Numeric settings for SET MARGIN, SET DECIMALS, SET MEMOWIDTH, SET TYPEAHEAD, SET ODOMETER, SET REFRESH, and SET REPROCESS
- The current directory
- ON KEY, ON ESCAPE, and ON ERROR settings
- SET ON/OFF command settings
- Programmable function key and SET FUNCTION settings

DISPLAY STATUS pauses when the results pane is full and displays a dialog box prompting you to display another screenful of information. Use the TO FILE clause to send the information to a file. Use the TO PRINTER clause to send the information to the printer. In either case, you can use SET CONSOLE OFF to suppress the display of the information in the results pane.

DISPLAY STATUS is the same as LIST STATUS, except that LIST STATUS does not pause with the first window of information but rather continuously lists the information until complete. This makes LIST STATUS more appropriate for outputting to a file or printer.

DISPLAY STRUCTURE

Topic group Related topics Example

Displays the field definitions of the specified table.

Syntax

DISPLAY STRUCTURE
[IN <alias>]

[TO FILE <filename> | ? <filename skeleton>]
[TO PRINTER]

IN <alias>

Identifies the work area of the open table whose structure you want to display rather than that of the current table. For more information, see Aliases.

TO FILE <filename> | ? | <filename skeleton>

Directs output to the text file <filename>, in addition to the results pane of the Command window. By default, *dBASE Plus* assigns a .TXT extension to <filename> and saves the file in the current directory. The ? and <filename skeleton> options display a dialog box in which you specify the name of the target file and the directory to save it in.

TO PRINTER

Directs output to the printer in addition to the results pane of the Command window.

Description

Use DISPLAY STRUCTURE to view the structure of the current or a specified table in the results pane of the Command window. DISPLAY STRUCTURE displays the following information about the current or specified table:

- Name of the table
- Type of table (Paradox, dBASE, or SQL)
- Table type version number
- Number of records
- Date of last update (DBF only)
- Fields
  - Field number
  - Field name (if SET FIELDS is ON, the greater-than symbol (>) appears next to each field specified with the SET FIELDS TO command)
  - Type
  - Length
  - Dec: The number of decimal places in a numeric or float field
  - Index: Whether there is a simple index on that field
- Number of bytes per record (the sum of field lengths; for DBF includes one additional byte reserved for storing the asterisk that marks a record as deleted)

Multiply the total number of bytes per record by the number of records in the table to estimate the size of a DBF table (excluding the size of the table header).

DISPLAY STRUCTURE pauses when the results pane is full and displays a dialog box prompting you to display another screenful of information. Use the TO FILE clause to send the information to a file. Use the TO PRINTER clause to send the information to the printer. In either case, you can use SET CONSOLE OFF to suppress the display of the information in the results pane.

DISPLAY STRUCTURE is the same as LIST STRUCTURE, except that LIST STRUCTURE does not pause with the first window of information but rather continuously lists the information until complete. This makes LIST STRUCTURE more appropriate for outputting to a file or printer.

The index column of DISPLAY STRUCTURE provides information about simple indexes only. When DISPLAY STRUCTURE indicates no index exists for a particular field (**N**), this does not preclude the possibility that it is included in an existing complex index. Complex indexes are those containing expressions such as "last_name+first_name".

Neither DISPLAY STRUCTURE nor LIST STRUCTURE permit modification of an existing table structure. To alter the structure, use MODIFY STRUCTURE.

HELP

Topic group Related topics Example

Activates the *dBASE Plus* Help system.

Syntax

HELP [<help topic>]

<help topic>

The Help topic you access with HELP.

Description

Use the HELP command in the Command window to get information on *dBASE Plus*.

dBASE Plus locates the first Help topic in the index beginning with <help topic>. If only one topic with the index entry is found, that topic is displayed. If there are multiple matches, Help displays a dialog box to let you choose the topic. If there is no match, the Help index is opened with <help topic> as the current search value.

Pressing F1 gives you context-sensitive help based on the control or window that currently has focus, or text that is highlighted in the Command window or Source Editor.

INSPECT( )

Topic group Related topics

Opens the Inspector, a window that lists object properties and lets you change their settings.

Syntax

INSPECT(<oRef>)

<oRef>

A reference to the object that you want to inspect.

Description

Use INSPECT( ) to examine and change object properties directly. For example, during program development you can use INSPECT( ) to evaluate objects and experiment with different property settings.

The Inspector is modeless, and doesn't affect program execution.

Note

You can access the Inspector from the Form designer by pressing F11.

You can get help on any property in the Inspector by selecting the property and pressing F1.

LIST...

Topic group Related topics

Lists information in the results pane of the Command window without pausing.

Syntax

LIST COVERAGE <filename1> | ? | <filename skeleton 1>
[ALL]
[SUMMARY]
[TO FILE <filename2> | ? | <filename skeleton 2>]
[TO PRINTER]

LIST MEMORY
[TO FILE <filename> | ? | <filename skeleton>]
[TO PRINTER]

LIST STATUS
[TO FILE <filename> | ? | <filename skeleton>]
[TO PRINTER]

LIST STRUCTURE
[IN <alias>]
[TO FILE <filename> | ? <filename skeleton>]
[TO PRINTER]

Description

The LIST commands listed above are the same as their DISPLAY command counterparts, except that LIST commands do not pause with the first window of information but rather continuously list the information until complete. This makes the LIST versions more appropriate for outputting to a file or printer.

MODIFY...

Topic group Related topics

Modifies the corresponding file.

Syntax

MODIFY COMMAND [<*filename*> | ? | <*filename skeleton*>]

MODIFY DATAMODULE [<*filename*> | ? | <*filename skeleton*>]

MODIFY FILE [<*filename*> | ? | <*filename skeleton*>]

MODIFY FORM [<*filename*> | ? | <*filename skeleton*>]

MODIFY LABEL [<*filename*> | ? | <*filename skeleton*>]

MODIFY MENU [<*filename*> | ? | <*filename skeleton*>]

MODIFY POPUP [<*filename*> | ? | <*filename skeleton*>]

MODIFY QUERY [<*filename*> | ? | <*filename skeleton*>]

MODIFY REPORT [<*filename*> | ? | <*filename skeleton*>]

Description

The MODIFY commands listed above operate the same as their CREATE command counterparts, except that if the specified file exists it is modified without prompting. For more information, see the corresponding CREATE commands.

MODIFY PROJECT

Topic group Related topics

Opens an existing project in the Project Explorer.

Syntax

MODIFY PROJECT [<filename> | ? | <*filename skeleton*>]

<filename> | ? | <filename skeleton>

The project file to open. The default extension is .PRJ. The ? and <filename skeleton> options display a dialog box from which you can select a file. If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory.

If you issue MODIFY PROJECT without an option, the Project Explorer is displayed, as if you issued CREATE PROJECT. Selecting File | New from the Main menu opens the Open Project dialog box from which you can navigate to your project (.prj) file

Description

MODIFY PROJECT opens the specified .PRJ file in the Project Explorer, making it the current project.

MODIFY STRUCTURE

Topic group Related topics Example

Allows you to modify the structure of the current table.

Syntax

MODIFY STRUCTURE

Description

Use MODIFY STRUCTURE to change the structure of the current table by adding or deleting fields, or changing a field name, width, or data type. Issuing the MODIFY STRUCTURE command opens the Table designer, an interactive environment in which you can create or modify the structure of a table. *dBASE Plus* will reopen a table in EXCLUSIVE mode if it wasn't already exclusive when you issued the MODIFY STRUCTURE command.

Before allowing changes to the structure of a dBASE table, *dBASE Plus* makes a backup of the original table assigning the file a .DBK extension. *dBASE Plus* then creates a new table file with the .DBF extension and copies the modified table structure to that file. When you've finished modifying a table structure, *dBASE Plus* copies the content of the backup file into the new structure. If data is accidentally truncated or lost, you can recover the original data from the .DBK file. Before modifying the structure of a table, make sure that you have sufficient disk space to create the backup file plus any temporary storage required to copy records between the two tables (approximately twice the size of the original table).

If a table contains a memo field, MODIFY STRUCTURE also creates a backup memo file to store the original memo field data. This file has the same name as the table, but is given a .TBK extension.

You shouldn't change a field name and its width or type at the same time. If you do, *dBASE Plus* won't be able to append data from the old field, and your new field will be blank. Change the name of a field, save the file, and then use MODIFY STRUCTURE again to change the field width or data type.

Also, don't insert or delete fields from a table and change field names at the same time. If you change field names, MODIFY STRUCTURE appends data from the old file by using the field position in the file. If you insert or delete fields as well as change field names, you change field positions and could lose data. You can, however, change field widths or data types at the same

time as you insert or delete fields. In those cases, since MODIFY STRUCTURE appends data by field name, the data will be appended correctly.

dBASE Plus successfully converts data between a number of field types. If you change field types, however, keep a backup copy of your original file, and check your new files to make sure the data has been converted correctly.

If you convert numeric fields to character fields, *dBASE Plus* converts numbers from the numeric fields to right-aligned character strings. If you convert a character field to a numeric field, *dBASE Plus* converts numeric characters in each record to digits until it encounters a non-numeric character. If the first character in a character field is a letter, the converted numeric field will contain zero.

You can convert logical fields to character fields, and vice versa. You can also convert character strings that are formatted as a date (for example, mm/dd/yy or mm-dd-yy) to a date field, or convert date fields to character fields. You can't convert logical fields to numeric fields.

In general, *dBASE Plus* attempts to make a conversion you request, but the conversion must be a sensible one or data may be lost. Numeric data can easily be handled as characters, but logical data, for example, cannot become numeric. To convert incompatible data types (such as logical to numeric), first add a new field to the file, use REPLACE to convert the data, then delete the old field.

If you modify the field name, length, or type of any fields that have an associated tag in the production (.MDX) file, the tag is rebuilt. If any indexes are open when you modify a table structure, *dBASE Plus* automatically closes those indexes when saving the modified table. You should re-index the table after you modify its structure.

SET

Topic group Related topics

Displays a dialog box for viewing and changing the values of many SET commands. The changed values are stored in the PLUS.ini file.

Syntax

SET

Description

Use SET to view and change settings interactively, instead of typing individual SET commands such as SET TALK ON in the Command window.

Note

Any changes you make to settings by using SET are automatically saved to PLUS.ini. This means that the settings will be in effect each time you start *dBASE Plus*. If you want to change the value of SET commands only temporarily, issue individual SET commands in the Command window or in a program.

Issuing SET is the same as choosing the Properties|Desktop menu option.

SET AUTONULLFIELDS

Topic group Related topics

Global setting used to affect the status of fields in blank records when APPENDing to a Level 7 database.

Syntax

SET AUTONULLFIELDS ON | off

Description

Use SET AUTONULLFIELDS to determine whether empty fields are assigned a NULL value, or when applicable, filled with spaces, zero or "false".

When AUTONULLFIELDS is ON (the default setting), *dBASE Plus* allows an empty field to assume a “null value”. Null values are those which are nonexistent or undefined. **Null is the absence of a value** and, therefore, different from a blank or zero value.

When AUTONULLFIELDS is OFF, numeric fields (long, float, etc.) are assigned a value of zero, logical fields a value of "false", and character fields are filled with spaces.

OODML

Use the rowset object's autonullFields( ) property. This property will override the global setting.

SET BELL

Topic group Related topics Example

Turns the computer bell on or off and sets the bell frequency and duration.

Syntax

SET BELL ON | off

SET BELL TO
[<frequency expN>, <duration expN>]

<frequency expN>

The frequency of the bell tone in cycles per second, which must be an integer from 37 to 32,767, inclusive.

<duration expN>

The duration of the bell tone in milliseconds, which must be an integer from 1 to 2000 (two seconds), inclusive.

Description

When SET BELL is ON, *dBASE Plus* produces a tone when you fill a data entry field or enter invalid data. SET BELL TO determines the frequency and duration of this tone, unless the computer is running Windows 95 and has a sound card. In that case, the Windows Default sound is played (through the sound card) instead of the tone.

Displaying CHR(7) in the results pane of the Command window sounds the "bell" whether SET BELL is ON or OFF.

SET BELL TO with no arguments sets the frequency and duration to the default values of 512 Hertz (cycles per second) for 50 milliseconds.

SET BLOCKSIZE

Topic group Related topics Example

Changes the default block size of memo field and .MDX index files.

Syntax

SET BLOCKSIZE TO <expN>

<expN>

A number from 1 to 63 that sets the size of blocks used in memo and .MDX index files. (The actual size in bytes is the number you specify multiplied by 512.)

Default

The default for SET BLOCKSIZE is 1 (for compatibility with dBASE III PLUS). To change the default, update the BLOCKSIZE setting in PLUS.ini.

Description

Use SET BLOCKSIZE to change the size of blocks in which *dBASE Plus* stores memo field files and .MDX index files on disk. The actual number of bytes used in blocks is <expN> multiplied by 512. Instead of using SET BLOCKSIZE, you can set the block size used for memo and .MDX index files individually, by using SET MBLOCK and SET IBLOCK commands.

After the block size is changed, memo fields created with the COPY, CREATE, and MODIFY STRUCTURE commands have the new block size. To change the block size of an existing memo field file, use the SET BLOCKSIZE command to change the block size and then copy the table containing the associated memo field to a new file. The new file then has the new block size.

SET COVERAGE

Topic group Related topics

Determines whether program files are compiled with coverage.

Syntax

SET COVERAGE on | OFF

Description

A coverage file is a binary file containing cumulative information on how many times, if any, *dBASE Plus* enters and exits (and thus fully executes) each logical block of a program. Use SET COVERAGE as a program development tool to determine which program lines *dBASE Plus* executes and doesn't execute each time you run a program.

A program file is either compiled with coverage or not. To disable coverage analysis, the file must be recompiled with coverage off.

There are two ways to control compilation with coverage. The first way is with SET COVERAGE, which can be either ON or OFF. The second way is with the coverage #pragma in the program file. The #pragma directive overrides the SET COVERAGE setting.

If a file is compiled with coverage enabled, *dBASE Plus* creates a new coverage file or updates an existing one. When *dBASE Plus* creates a coverage file, it uses the name of the program file, and changes the last letter of the extension to the letter "V"; unless the file is a .PRG, in which case the coverage file has an extension of .COV. For example, the coverage file for GRADES.PRG is GRADES.COV, and the coverage file for STUDENTS.WFM is STUDENT.WFV.

To view the contents of a coverage file, use DISPLAY COVERAGE or LIST COVERAGE. If the coverage file reveals that some lines aren't executing, you can respond by changing the program or the input to the program to make the lines execute. In this way, you can make sure that you test all lines of code in the program.

Coverage analysis divides a program into logical blocks. A logical block doesn't include commented lines or programming construct command lines such as IF and ENDIF. It does, however, include command lines within programming construct command lines. If your program doesn't contain any programming constructs (like IF, DO WHILE, FOR...ENDFOR, SCAN...ENDSCAN, LOOP, DO CASE, DO...UNTIL), the program has only one logical block consisting of all uncommented command lines.

The coverage file identifies a logical block by its corresponding program line number(s):

```
Line 1 * UPDATES.PRG
Line 2 SET TALK OFF  Block 1 (Lines 2-3)
Line 3 USE Customer INDEX Salespers
Line 4 SCAN
Line 5 DO CASE
Line 6 CASE Salesper = "S-12"
Line 7 SELECT 2  Block 2 (Lines 7-8)
Line 8 USE S12
Line 9 CASE Salesper = "L-5"
Line 10 SELECT 2  Block 3 (Lines 10-11)
Line 11 USE L5
Line 12 CASE Salesper = "J-25"
Line 13 SELECT 2  Block 4 (Lines 13-14)
Line 14 USE J25
Line 15 ENDCASE
Line 16 DO Changes  Block 5 (Lines 16-17)
Line 17 SELECT 1
Line 18 ENDSCAN
Line 19 CLOSE ALL    Block 6 (Lines 19-20)
Line 20 SET TALK ON
```

dBASE Plus writes the coverage file to disk when the program is unloaded from memory or when you issue a LIST COVERAGE or DISPLAY COVERAGE. To unload a program from memory, use CLEAR PROGRAM.

SET DESIGN

Topic group Related topics Example

Determines whether CREATE and MODIFY commands can be executed.

Syntax

SET DESIGN ON | off

Default

The default for SET DESIGN is ON. To change the default, set the DESIGN parameter in PLUS.ini. To do so, either use the SET command to specify the setting interactively, or enter the DESIGN parameter directly in PLUS.ini.

Description

When SET DESIGN is ON, *dBASE Plus* lets you use CREATE and MODIFY commands to create and modify tables, forms, labels, reports, text, and queries. To prevent users of your applications from creating and modifying these types of files, issue SET DESIGN OFF in your programs.

If you issue SET DESIGN ON or OFF in a subroutine, the setting is effective only during execution of that subroutine.

SET DEVELOPMENT

Topic group Related topics

Determines whether *dBASE Plus* automatically compiles a program, procedure, or format file when you change the file and then execute it or open it for execution.

Syntax

SET DEVELOPMENT ON | off

Default

The default for SET DEVELOPMENT is ON. To change the default, set the DEVELOPMENT parameter in PLUS.ini. To do so, either use the SET command to specify the "Ensure Compilation" setting interactively, or enter the DEVELOPMENT parameter directly in PLUS.ini.

Description

When SET DEVELOPMENT is ON and you execute a program file with DO, or open a procedure or format file, *dBASE Plus* compares the time and date stamp of the source file and the compiled file. If the source file has a later time and date stamp than the compiled file, *dBASE Plus* recompiles the file.

When SET DEVELOPMENT is ON and you change a source program, procedure, or format file with MODIFY COMMAND, *dBASE Plus* erases the corresponding compiled file. When you then execute the program or open the procedure or format file, *dBASE Plus* recompiles it.

When SET DEVELOPMENT is OFF, *dBASE Plus* doesn't compare time and date stamps, and executes or opens existing compiled program, procedure, or format files. When you modify a source file and then open or execute it, *dBASE Plus* first looks for a compiled file in memory and executes it if found. If no compiled file is in memory, *dBASE Plus* looks for a compiled disk file and executes it if found. If no compiled file is found, *dBASE Plus* compiles the file.

When you DO a program, open a procedure file with SET PROCEDURE, or open a format file with SET FORMAT, *dBASE Plus* always looks for, opens, and executes a compiled file. Therefore, if *dBASE Plus* can't find a compiled version of a source file when you execute or open the source, *dBASE Plus* compiles the file regardless of the SET DEVELOPMENT setting.

During program development, when you're editing files often, you should turn SET DEVELOPMENT ON. This ensures that you're always executing an up-to-date compiled file.

Turn SET DEVELOPMENT OFF when you no longer plan to change any source code. Turning SET DEVELOPMENT OFF speeds up program execution because *dBASE Plus* doesn't have to check time and date stamps. You might want to set the DEVELOPMENT parameter to OFF in the PLUS.ini file you distribute with your compiled code.

SET ECHO

Topic group

Opens the *dBASE Plus* Debugger. This command is supported primarily for compatibility with dBASEIV. In *dBASE Plus*, use DEBUG to open the debugger.

Syntax

SET ECHO on | OFF

The default for SET ECHO is OFF.

Description

Use SET ECHO to turn on the Debugger and view or control program execution interactively. SET ECHO is identical to DEBUG. For more information, see DEBUG.

SET EDITOR

Topic group Related topics Example

Specifies the text editor to use when creating and editing programs and text files.

Syntax

SET EDITOR TO
[<expC>]

<expC>

The expression you would enter at the DOS prompt or as the Windows command line to start the editor, usually the name of the editor's executable file (.EXE) or a Windows .PIF file. If <expC> doesn't include the file's full path name, *dBASE Plus* looks for the file in the current directory, then in the DOS path.

Default

The default for SET EDITOR is the *dBASE Plus* internal Source editor. To specify a different default editor, set the EDITOR parameter in PLUS.ini. To do so, either use the SET command to specify the setting interactively, or enter the EDITOR parameter directly in PLUS.ini.

Description

Use SET EDITOR to specify an editor other than the default *dBASE Plus* Source editor to use when creating or editing text files. The file name you specify can be any text editor that produces standard ASCII text files. The specified editor opens when you issue CREATE/MODIFY FILE or CREATE/MODIFY COMMAND. If you issue SET EDITOR TO without a file name for <expC>, *dBASE Plus* returns to the default editor.

You can use SET EDITOR to specify a .PIF file, which is a Windows file that controls the Windows environment for a DOS application, or a Windows .EXE file. Start the DOS editor by running the .PIF file rather than the .EXE. For more information about .PIF files, see your Windows documentation. If there is not enough memory available to access an external editor, *dBASE Plus* returns an "Unable to execute DOS" error message.

If the text editor you specify is already in use when you open a memo or file for editing, a second instance of the editor starts.

SET HELP

Topic group Related topics Example

Determines which Help file (.HLP) the *dBASE Plus* Help system uses.

Syntax

SET HELP TO

[<help filename> | ? | <help filename skeleton>]

<help filename> | ? | <help filename skeleton>

Identifies the Help file to activate. ? and <filename skeleton> display a dialog box, from which you can select a file. If you specify a file without including its extension, *dBASE Plus* assumes .HLP.

Description

Use SET HELP TO to specify which Help file to use when the *dBASE Plus* Help system is activated.

The Help file is opened automatically when you start *dBASE Plus* if you place the file in the *dBASE Plus* home directory. SET HELP TO closes any open Help file before it opens a new file.

SET IBLOCK

Topic group Related topics Example

Changes the default block size used for new .MDX files.

Syntax

SET IBLOCK TO <expN>

<expN>

A number from 1 to 63 that sets the size of index blocks allocated to new .MDX files. The default value is 1. (The actual size in bytes is the number you specify multiplied by 512 bytes; however, the minimum size of a block is 1024 bytes.) To change the default, update the IBLOCK setting in PLUS.ini. To do so, either use the SET command to specify the setting interactively, or enter the IBLOCK parameter directly in PLUS.ini.

Description

Use SET IBLOCK to change the size of blocks in which *dBASE Plus* stores .MDX files on disk to improve the performance and efficiency of indexes. You can specify a block size from 1024 bytes to approximately 32K. The IBLOCK setting overrides any previous block size defined by the SET BLOCKSIZE command or specified in the PLUS.ini file. After the block size has been changed, new .MDX index files are created with the new block size.

Multiple index (.MDX) files are composed of individual index blocks (or nodes). Nodes contain the value of keys corresponding to individual records and provide the information to locate the appropriate record for each key value. Since the IBLOCK setting determines the size of nodes, the setting also determines the number of key values that can fit into each node. When a single node can't contain all the key values in an index, *dBASE Plus* creates one or more parent nodes. These intermediate nodes also contain key values. Instead of pointing to record numbers, however, intermediate nodes point to leaf nodes or other lower-level intermediate nodes. If you increase the size of index blocks and create a new .MDX file, the new and larger leaf nodes contain more key values.

Whether you can improve performance by storing key values in larger or smaller nodes depends on several factors: the distribution of data, if tables are linked together, the length of key values and the type of operation requested. Typically, every .MDX file contains more than one index tag. Finding the best setting for a given .MDX file requires experimentation because the best size for one index tag might not be the best size for another.

The following is a list of basic principles governing index performance.

- Since nodes might not be sequential, *dBASE Plus* reads only one node at a time from the disk. Reading more than one node is usually inefficient, because typically the second node is not the next node in the sequential list.
- Once a node is read into memory, *dBASE Plus* attempts to store it there for later use.

When users link several tables together, for example, with SET RELATION, performance is better if all the relevant nodes for the tables are in memory simultaneously. For example, if a large node for table B pushes out the previously read node for table A, *dBASE Plus* must find and read the table A node again from disk when the node for table A needs to be used again. If both nodes remain in memory, performance can be improved.

When tables have many identical key values, *dBASE Plus* might have to store them in many nodes. In this situation, performance might be improved by increasing the node size so that *dBASE Plus* reads fewer nodes from disk to load the same number of key values into memory.

Small node sizes can cause performance degradation. This occurs because as nodes are read in and out, *dBASE Plus* attempts to cache them all. When the small nodes are removed from memory by more recently read nodes, they leave unused spaces in memory that are too small to contain larger nodes. Over time, memory can become fragmented, resulting in slower performance.

SET MBLOCK

Topic group Related topics Example

Changes the default block size of new memo field (.DBT) files.

Syntax

SET MBLOCK TO <expN>

<expN>

A number from 1 to 512 that sets the size of blocks used to store new memo (.DBT) files. (The actual size in bytes is the number you specify multiplied by 64.)

Default

The default value for SET MBLOCK is 8 (or 512 bytes). To change the default, update the MBLOCK setting in PLUS.ini. To do so, either use the SET command to specify the setting interactively, or enter the MBLOCK parameter directly in PLUS.ini.

Description

Use SET MBLOCK to change the size of blocks in which *dBASE Plus* stores new memo field (.DBT) files on disk. You can specify a block size from 64 bytes to approximately 32K. The MBLOCK setting overrides any previous block size defined by the SET BLOCKSIZE command or specified in the PLUS.ini file. After the block size has been changed, new memo .DBT files are created with the new block size. *dBASE Plus* stores data in each memo field in a group made up of as many blocks as needed.

After the block size is changed, memo fields created with the COPY, CREATE, and MODIFY STRUCTURE commands have the new block size. To change the block size of an existing memo field file, use the SET BLOCKSIZE command to change the block size and then copy the table containing the associated memo field to a new file. The new file then has the new block size.

When the block sizes are large and the memo contents are small, memo (.DBT) files contain unused space and become larger than necessary. If you expect the contents of the memo fields to occupy less than 512 bytes (the default size allocated), set the block size to a smaller size to reduce wasted space. If you expect to store larger pieces of information in memo fields, increase the size of the block.

SET MBLOCK is similar to the older SET BLOCKSIZE command except for two advantages:

You can allocate different block sizes for memo field and index data, whereas SET BLOCKSIZE requires the same block size for both. To allocate block sizes for index data, use SET IBLOCK.

You can specify smaller blocks with SET MBLOCK than with SET BLOCKSIZE. SET BLOCKSIZE creates blocks in increments of 512 bytes, compared to 64 bytes with SET MBLOCK.

SET STEP

Topic group

SET STEP ON opens the *dBASE Plus* Debugger. This command is supported primarily for compatibility with dBASEIV. In *dBASE Plus*, use DEBUG to open the debugger.

Syntax

SET STEP on | OFF

The default for SET STEP is OFF.

Description

Use SET STEP to turn on the Debugger and view or control program execution interactively. SET STEP is identical to DEBUG. For more information, see DEBUG.

SET TALK

Topic group Related topics Example

Determines whether *dBASE Plus* displays messages in the status bar, or displays memory variable assignments in the results pane of the Command window.

Syntax

SET TALK ON | off

Default

The default for SET TALK is ON. To change the default, set the TALK parameter in PLUS.ini. To do so, either use the SET command to specify the setting interactively, or enter the TALK parameter directly in PLUS.ini.

Description

When SET TALK is ON, *dBASE Plus* uses the current SET ODOMETER setting to indicate when operations such as COUNT and SORT are in progress in the status bar. It also displays the results of memory variable assignments (using STORE or =) in the results pane of the Command window.

Depending on the amount of memory your system has and the amount of memory particular operations require, issuing SET TALK OFF might improve the performance of some operations.

Use SET TALK with SET ALTERNATE to send SET TALK output to a file or printer rather than to the results pane of the Command window.

When SET TALK is ON, *dBASE Plus* reports the results of the BUILD command in a dialog box. If SET TALK is OFF, nothing happens when BUILD is successful.

Application Shell

Appshell

Topic group

This section covers supporting application elements such as menus, popups, toolbars, standard dialogs, keyboard behavior, and the *_app* object.

_app

Topic group Related topics Example

The global object representing the currently running instance of *dBASE Plus*.

Syntax

The _app object is automatically created when you start *dBASE Plus*.

Properties

The following tables list the properties, events and methods of the _app object.

Property	Default	Description
allowDEOExeOverride	true	Whether a dBASE Plus application checks for external objects
allowYieldOnMsg	false	Whether dBASE Plus checks for messages during progam execution
allUsersPath		Full path to the shared folder where shared files or folders may be located for dBASE Plus or for an application
baseClassName	APPLICATION	Identifies the object as an instance of the dBASE Plus application
charSet		The current global character set
className	(APPLICATION)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
currentUserPath		Full path to the user's private dBASE or application folder.
databases	Object array	An array containing references to all database objects used by the Navigator
ddeServiceName	DBASE	The name used to identify each instance of dBASE Plus when used as a DDE service
detailNavigationOverride	0 - Use rowset's detail settings	Controls whether or not a rowset's *navigateMaster* and *navigateByMaster* properties are overridden
errorAction	4 - Show Error Dialog	Default action to be taken when an error is encountered
errorHTMFile	error.htm	Filename of an HTM file template (runtime web apps only)
errorLogFile	PLUSErr.log	Filename of the error log file to be used when the _app object's *errorAction* property is set to 2,3, or 5.
errorLogMaxSize	100	Approximate maximum size of the error log file (kilobytes)
errorTrapFilter	0 - Trap all errors	Enables, or disables, the detection of certain kinds of errors.
exeName		Drive, path and filename of the currently running instance of PLUS.exe or a dBASE Plus application .exe.
frameWin	Object	The dBASE Plus MDI frame window
iniFile		Full path and filename to the dBASE or Application .ini file
insert	true	Whether text typed at the cursor is inserted or overwrites existing text
language		The currently used display language in the design and runtime environments. Read only.
lDriver		The current global language driver
printer	Object	Configuration properties for the default printer
roamingUsersPath		Full path the the current users' roaming folder where you can choose to store data for an application that will roam from one workstation to another if a network hosting an application is configured to support roaming users

session	Object	The default Session object
sourceAliases	Object array	A read-only array containing references to all Source Aliases defined in the PLUS.ini.
speedBar	true	Whether to display the default toolbar.
statusBar	true	Whether to display the status bar at the bottom of the MDI frame window
terminateTimerInterval	5000 milliseconds	Determines the amount of time, in milliseconds, between the closing of an application .exe and the removal of PLUSrun.exe from the Web servers memory
useUACPaths		Indicates whether or not dBASE or a dBASE application should create and maintain private copies of various files and folders for each user according to Window's User Account Control (UAC) rules.
web	false	Indicates whether an application .exe was built using the WEB parameter

Event	Parameters	Description
beforeRelease		before an object is released from memory.
onInitiate	<topic expC>	When a client application requests a DDE link with dBASE Plus as the server, and no DDETopic object for the specified topic exists in memory.

Method	Parameters	Description
addToMRU( )	<filename> <launchMode>	When called, adds a file to the “most recently used” files list located on the “Files | Recent Files” and “Files | Recent Projects” menus.
executeMessages( )		When called, handles pending messages during execution of a processing routine.
themeState( )		Indicates whether themes are in use for the application.

Description

Use _app to control and get information about the currently running instance of *dBASE Plus.* The insert property controls the insert or overwrite behavior of typed text in all forms, the Source Editor, and the Command window. It is toggled by pressing the Insert key. You may show or hide the default toolbars and the status bar. To control other aspects of the main application window, use the *_app.frameWin* object.

The databases array contains references to all databases opened by the Navigator. The default database is the first element in that array. The session property points to the default session. Therefore _app.databases[1].session and _app.session point to the same object.

To use *dBASE Plus* as a DDE server, set the *ddeServiceName* to a unique identifier if there is more than one instance of *dBASE Plus* running or if you want your application to have a specific DDE service name other than the default "DBASE", then assign an *onInitiate* event handler to handle the service request.

The _app object is also used to store important global values and other objects used by your application. Dynamically creating properties of _app is preferable to creating public variables.

Variables may be inadvertently released or conflict with other variable names.

Objects referenced only in variables cannot communicate with each other using object-oriented techniques. Objects attached to the same parent object, in this case _app, can.

_app.frameWin

Topic group Related topics Example

The *dBASE Plus* MDI frame window.

Syntax

The *_app.frameWin* object is automatically created when you start *dBASE Plus*.

Properties

The following table lists the properties and methods of the *_app.frameWin* object. (No events are associated with the *_app.frameWin* object.)

Property	Default	Description
baseClassName	FRAMEWINDOW	Identifies the object as an instance of an MDI frame window
className	(FRAMEWINDOW)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
hWnd		The Windows handle to the frame window
text	dBASE Plus	The title displayed in the frame window
systemTheme	true	Whethor or not to use the common controls in the manifest file for Windows XP or Windows Vista.
visible	true	Whether the frame window is visible
windowState		The state of the frame window: 0=Normal, 1=Minimized, 2=Maximized

Event	Parameters	Description
onClose()		After the framewin has been closed.

Method	Parameters	Description
hasHScrollBar( )		Indicates whether a frame window uses a horizontal scrollbar.
hasVScrollBar( )		Indicates whether a frame window uses a vertical scrollbar.

Description

_app.frameWin represents the main *dBASE Plus* application window. This window is the frame window that contains all MDI windows during development and in an MDI application. If your application uses SDI windows only, the MDI frame window is usually hidden with the SHELL( ) function.

If you assign a MenuBar to *_app.frameWin*, that menu becomes the default menu that is visible when no MDI windows are open, or the current MDI window has no menu of its own. If you are using SHELL( ) to hide the Navigator and Command window in an MDI application, you must call SHELL( ) after assigning the *_app.frameWin* menu.

class Menu

Topic group Related topics Example

A menu item in a menubar or popup menu.

Syntax

[<oRef> =] new Menu(<parent>)

<oRef>

A variable or property—typically of <parent>—in which to store a reference to the newly created Menu object.

<parent>

The parent object—a MenuBar, Popup, or another Menu object—that contains the Menu object.

Properties

The following tables list the properties, events, and methods of the Menu class.

Property	Default	Description
baseClassName	MENU	Identifies the object as an instance of the Menu class.
before		The next sibling menu object
checked	false	Whether to display a checkmark next to a menu item.
checkedBitmap		An image to represent the mark to appear next to the menu item if checked is true.
className	(MENU)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
enabled	true	Determines if the menu can be selected.
helpFile		Help file name
helpId		Help index topic or context number for context-sensitive help
id	-1	Supplementary ID number for menu item
name		The name of the menu item.
parent		The menu item's immediate container
separator	false	Whether the menu object is a separator instead of a selectable menu item.
shortCut		The key combination that fires the onClick event.
statusMessage		The message to display on the status bar when the menu item has focus.
text		The text of the menu item prompt.
uncheckedBitmap		An image to represent the mark to appear next to the menu item if checked is false.

Event	Parameters	Description
beforeRelease		before an object is released from memory.
onClick		After the menu item is chosen.
onHelp		When F1 is pressed—overrides context-sensitive help

Method	Parameters	Description
release( )		Explicitly removes the menu object from memory.

Description

Menu objects represent the individual menu items, or prompts, in a menu system. They can be attached to MenuBar objects, Popup objects, or other Menu objects so that:

- When attached to a menubar, they are the top-level menu items, such as the standard File and Edit menus.
- Menu items attached to a top-level menu item form the drop-down menu, such as the standard Cut and Paste menu items in the top-level Edit menu.
- Menu items attached to a popup comprise the items in the popup.

Any other menu items that have menu items attached to them become cascading menus.

Unless a menu item has other menu items attached (making it a cascading menu), selecting the menu item fires its *onClick* event. Actions are assigned to each menu item by creating an *onClick* event handler for the menu object.

The actions for the standard Undo, Cut, Copy, and Paste menu items and the Window menu are handled by assigning the menu items to the menubar's *editUndoMenu*, *editCutMenu*, *editCopyMenu*, *editPasteMenu*, and *windowMenu* properties respectively.

Menu objects are also used as separators in a drop-down or popup menu by setting their *separator* property to *true*. In this case, the menu item serves no other purpose and cannot be a cascading menu or have an *onClick* event handler.

Creating accelerators and pick characters

There are two ways to let the user choose a menu item by using the keyboard (which may be used at the same time):

Assign a key combination to the menu item's *shortCut* property. This is sometimes called an accelerator. For example, Ctrl+C is usually used for the Cut menu item. Pressing the accelerator chooses the menu item even if the menu item is not visible.

Specify a pick character in the *text* prompt of the menu item by preceding it with an ampersand (&). Pick characters work only when the menu item is visible. For top-level items in a menubar, you must press Alt and the pick character to activate the menu. Once the menu system is activated, pressing Alt in combination with the pick character is optional.

Note

Assigning F1 as the *shortCut* key for a menu item disables the built-in context-sensitive help based on the *helpFile* and *helpId* properties. The *onClick* for the menu item will be called instead. Therefore, if you have a menu item for Help it should not have F1 assigned as its *shortCut* key unless you want to handle help yourself.

class MenuBar

Topic group Related topics Example

A form's menu.

Syntax

[<oRef> =] new MenuBar(<parent> [, <name expC>])

<oRef>

A variable or property in which to store a reference to the newly created MenuBar object.

<parent>

The form (or the *_app.frameWin* object) to which you're binding the MenuBar object.

<name expC>

An optional name for the MenuBar object. The Menu Designer always uses the name "root". If not specified, the MenuBar class will auto-generate a name for the object.

Properties

The following table lists the properties of the Menubar class.

Property	Default	Description
baseClassName	MENUBAR	Identifies the object as an instance of the MenuBar class.

className	(MENUBAR)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
editCopyMenu		A menu item that copies selected text from a control to the Windows clipboard.
editCutMenu		A menu item that deletes selected text from a control and copies it to the Windows clipboard.
editPasteMenu		A menu item that pastes text from the Windows clipboard to the edit control with focus.
editUndoMenu		A menu item that restores the form to the state before the last edit operation was performed.
id	-1	A supplementary ID number for the object
name		The menubar object's name.
parent		An object reference that points to the parent form.
windowMenu		A top-level menu that lists open MDI windows.

Event	Parameters	Description
onInitMenu		When the menu is opened.

Method	Parameters	Description
release( )		Explicitly removes the menubar object from memory.

Description

A MenuBar object represents the top-level menu for a form. It contains one or more Menu objects which comprise the individual top-level menu items. The top-level menu of a form is displayed at the top of the form if the form's *mdi* property is *false*, or in the MDI frame window if the form's *mdi* property is true when the form has focus.

You may also designate a menubar that appears in the MDI frame when no MDI forms have focus by assigning a menubar to the *_app.frameWin* object.

The MenuBar object automatically binds itself to the <parent> form. Unlike other controls, you usually do not assign the resulting <oRef> as a property of the form, since this is done automatically, using the <name expC> that is specified. The Menu Designer always uses the name "root", so a form's menu is referred to with the object reference:

```
form.root
```

class Popup

Topic group Related topics

A popup menu assigned to a form.

Syntax

[<oRef> =] new Popup(<parent> [, <name expC>])

<oRef>

A variable or property in which to store a reference to the newly created Popup object.

<parent>

The form to which you're binding the Popup object.

<name expC>

An optional name for the Popup object. If not specified, the Popup class will auto-generate a name for the object.

Properties

The following tables list the properties, events and methods of the Popup class.

Property	Default	Description
alignment	Align Center	Aligns the popup menu horizontally relative to the right-click location (0=Align Center, 1=Align Left, 2=Align Right).
baseClassName	POPUP	Identifies the object as an instance of the Popup class.
className	(POPUP)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
id	-1	A supplementary ID number for the object
left		Sets the position of the left border.
name		The popup object's name.
parent		An object reference that points to the parent form.
top		Sets the position of the top border.
trackRight	true	Determines whether the popup menu responds to a right mouse click for selection of a menu item.

Event	Parameters	Description
beforeRelease		before an object is released from memory.
onInitMenu		After the popup menu is opened.

Method	Parameters	Description
open( )		Opens the popup menu.
release( )		Explicitly removes the popup object from memory.

Description

A Popup object represents a context or popup menu that is displayed when you right-click a form. You may also open the popup explicitly by calling its *open*( ) method.

A form may have any number of popup menu bound to it. Only one menu at time can be assigned to the form's *popupMenu* property; that is the menu that appears when right-clicking the form.

The popup contains Menu objects that represent the menu items in the popup.

class ToolBar

Topic group Related topics Example

A toolbar assigned to a form.

Properties

The following tables list the properties, events and methods of the ToolBar class.

Property	Default	Description
alignment	0	Set the alignment to determine the placement of the toolbar in the frame. Options are: 0 – Top //default 1 – left 2 – Right 3 – Bottom
allowdocking	3	Options for allowing the toolbar to be docked. Options are: 0 – None 1 – Horizontal 2 – Vertical 3 – Any //default
baseClassName	TOOLBAR	Identifies the object as an instance of the ToolBar class.
className	(TOOLBAR)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
enabled	true	Logical value which toggles the appearance of buttons on the toolbar between always raised (false) to only raised when the pointer is over a button (true).
flat	true	Logical value which toggles the appearance of buttons on the toolbar between always raised (false) to only raised when the pointer is over a button (true).
floating	false	Logical value that lets you specify your toolbar as docked (false) or floating (true).
form	null	Returns the object reference of the form to which the toolbar is attached.
frame		A reference to the FrameWindow object
gripper	true	Whether or not a gripper is attached to the left side of the toolbar. A gripper will allow the user to grab the toolbar and move it around
hWnd	0	Returns the toolbar's handle.

id		a Numeric ID given to the toolbar. The default is -1 until the toolbar is attached to a form. Then this number is set to 1000 initially for the first new toolbar that is created and attached to an open form.
		For each additional toolbar that is created and then attached to a form, it's ID is incremented by 1.
		
		For example in the command window you will find the following:
		t1 = new toolbar()
		?t1.id //returns -1
		f1 = new form()
		t1.attach(f1)
		?t1.id //returns -1
		f1.open()
		?t1.id //returns 1000
		t2 = new toolbar()
		f2 = new form()
		t2.attach(f2)
		f2.open()
		?t2.id //returns 1001

imageSource		can be used to specify an image containing multiple toolbutton images within it.

		In each toolbutton you must also set imageId to an image ID which is just the position of the desired image within the large image beginning with 0 for the left most image and increasing by 1 for each additional image.
		
		For example:
		t = new toolbar()
		t.imageSource = "RESOURCE #430 C:\ashley\EXE\Plus_EN.dll" // bitmap with multiple images
		t.imageWidth = 16 // 24 or 32 pixel depth each button image
		t.imageHeight = 16 // usually the same as imageWidth
		
		tb1 = new toolbutton(t)
		tb1.imageId = 0 // leftmost image from toolbar's imageSource
		
		tb2 = new toolbutton(t)
		tb2.imageId = 1 // use second from left image
		
		tb3 = new toolbutton(t)
		tb3.imageid = 2 // use third from left image
		
		etc...

imageHeight	0	Adjusts the default height for all buttons on the toolbar. Since all buttons must have the same height, if ImageHeight is set to 0, all buttons will match the height of the tallest button. If *ImageHeight* is set to a non-zero positive number, images assigned to buttons are either padded (by adding to the button frame) or truncated (by removing pixels from the center of the image or by clipping the edge of the image).
imageWidth	0	Specifies the width, in pixels, for all buttons on the toolbar.
left	0	Specifies the distance from the left side of the screen to the edge of a floating toolbar.
name		The name of the toolbar. You can name your toolbar to make identifying it a little easier, especially if you have multiple versions of your toolbar in memory.
padding	6	Pads the height of the toolbar.
parent		The current FRAMEWINDOW object.
text		String that appears in the title bar of a floating toolBar.
toolItems		A read only property which contains an array object containing the toolbutton objects that are defined in the toolbar. For example: t = new toolbar() tb = new toolButton(t) ?t.toolItems[1].baseClassName //returns TOOLBUTTON or ?t.toolItems["tb"].baseClassName
top	0	Specifies the distance from the top of the screen to the top of a floating toolbar.
transparent	false	whether or not the background of the toolbar shows the same as the frame's background.
visible	true	Logical property that lets you hide or reveal the toolbar.

Event	**Parameters**	**Description**
onCommand		Fires when any toolbutton is pressed. Can be used to

		determine what happens when any toolbutton is pressed by the user. This example checks to see if the current database is the correct one and based on that enables or disables the onClick event for a list of toolbuttons: app.t = new toolbar() _app.t.onCommand = {;if upper(app.databases.current.databasename) <> "DBASESAMPLES"; this.b1.onClick = null; this.b2.onClick = null; else; this.b1.onClick = CLASS::B1_ONCLICK; this.b2.onClick = CLASS::B2_ONCLICK; endif; ?"onclick = ",this.b1.onclick}
onUpdate		Fires repeatedly while application is idle to update the status of the toolbuttons

Method	Parameters	Description
attach( )	<form>	Establishes link between the toolbar and a form
detach( )	<form>	Breaks link between the toolbar and a form

Description

Use class ToolBar to add a toolbar to a form.

class ToolButton

Topic group Related topics Example

Defines the buttons on a toolbar.

Properties

The following tables list the properties and events of the ToolButton class. (No methods are associated with this class.)

Property	Default	Description
baseClassName	TOOLBUTTON	Identifies the object as an instance of the ToolButton class.
bitmap		Graphic file (any supported format) or resource reference that contains one or more images that are to appear on the button.
bitmapOffset	0	Specifies the distance, in pixels, from the left of the specified Bitmap to the point at which your button graphic begins. This property is only needed when you specify a Bitmap that contain a series of images arranged from left to right. Use with *BitmapWidth* to specify how many pixels to display from the multiple-image Bitmap. Default is 0 (first item in a multiple-image Bitmap).
bitmapWidth	0	Specifies the number of pixels from the specified Bitmap that you want to display on your button. This property is only needed when you specify a Bitmap that contain a series of images arranged from left to right. Use with *BitmapOffset*, which specifies the starting point of the image you want to display.
checked	false	Returns true if the button has its*TwoState* property set to true. Otherwise returns false.
className	(TOOLBUTTON)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
enabled	true	Logical value that specifies whether or not the button responds when clicked. When set to false, the operating system attempts to visually change the button with hatching or

		a low-contrast version of the bitmap to indicate that the button is not available.
parent	N/A	An object reference that points to the parent ToolBar.
separator	false	Logical value that lets you set a vertical line on the toolbar to visually group buttons. If you specify a separator button, only its Visible property has any meaning.
speedTip		Specifies the text that appears when the mouse rests over a button for more than one second.
twoState	true	Logical value that determines whether the button displays differently when it has been depressed and consequently sets the *Checked* property to true.
visible	false	Logical value that lets you hide (*false*) or show (*true*) the button.

Event	Parameters	Description
onClick		After the button is clicked.

Description

Use class ToolButton to define the buttons on an existing toolbar.

addToMRU( )

Topic group

Use the *addToMRU*( ) method to add a file to the “most recently used” files list located on the “Files | Recent Files” and “Files | Recent Projects” menus.

Syntax

_app.addToMRU(<filename>, <launchMode>)

<filename>

File name and optional path or alias

In order to be added to the Recent Files list:

If no alias is specified, a file must exist and must not have an extension of .TMP

If an alias IS specified, it is added to the list without first checking if it exists

In order to be added to the Recent Projects list:

A file must have an extension of .PRJ and <launchMode> must be 4 - "Run"

<launchMode>

A number from 0 to 12 specifying how the file should be launched, or opened, when a user selects it from the "Recent Files" list.

Number	Launch Mode Description
0	Use the default method based on files extension
1	Launch as if user selected, "File | New", from menu
2	reserved
3	Open file in appropriate designer
4	"Run" the file for Projects: Open in Project Explorer

		for Programs: DO <program> for Tables: Edit Records for Queries: Edit Records
5	Alternate "Run"	for Programs: DEBUG for Tables: APPEND for Queries: SET VIEW only for .qry: SET FILT only
6	Open a table via USE	
7	reserved	
8	Close	
9	Open in Source Editor	
10	Compile	
11	Debug	
12	Build	

Property of

_app object

Description

The *addToMRU*( ) method can be called from a dBL program that runs in the *dBASE Plus* IDE. If called in a *dBASE Plus* runtime application, it will RETURN without doing anything.

allowDEOExeOverride

Topic group Related topics

Determines whether an application will search for external objects.

Property of

_app object

Description

Dynamic External Objects (DEO) is active in *dBASE Plus* by default, which allows applications to perform an automatic search for external objects. Setting the *allowDEOExeOverride* property to false prevents procedures, built into an application .exe, from being overridden by external objects, by first searching within the application .exe. Only procedures that do NOT exist within the .exe are then searched for outside of the application .exe.

When *allowDEOExeOverride* is false:

dBASE Plus searches for objects as follows:

1. It looks inside the application's .exe file, the way Visual dBASE did in the past.
2. It looks in the "home" folder from which the application was launched.
3. It looks in the .ini file to see if there's a series of search paths specified. It checks all the paths in the list looking for the object file requested by the application.

When *allowDEOExeOverride* is true (the default):

dBASE Plus searches for objects as follows:

4. It looks in the "home" folder from which the application was launched.
5. It looks in the .ini file to see if there's a series of search paths specified. It checks all the paths in the list looking for the object file requested by the application.
6. It looks inside the application's .exe file, the way Visual dBASE did in the past.

The default setting for the *allowDEOExeOverride* property is true.

allowYieldOnMsg

Topic group

Enables or disables the message pump during execution of *dBASE Plus* applications.

Property of

_app object

Description

Use *allowYieldOnMsg* to allow *dBASE Plus* to be more responsive while the interpreter is running a lengthy routine.

When *allowYieldOnMsg* is set to *false*, the default setting, *dBASE Plus* does NOT check for messages until the interpreter completes running the current program. Setting *allowYieldOnMsg* to *false* will speed up processing.

When *allowYieldOnMsg* is set to *true*, the *dBASE Plus* interpreter periodically checks for messages waiting to be processed and, if found, sends them to the appropriate routines.

Messages pumped when *allowYieldOnMsg* is set to *true* include the following:

WM_PAINT - signals *dBASE Plus* to repaint a control, container, form, or framewindow to update its contents.

WM_CLOSE - signals *dBASE Plus* to close a window.

WM_QUERYENDSESSION - signals that the current user is logging out of Win 9x, Win NT, Win ME or 2000, which in turn, causes *dBASE Plus* to shut down.

WM_QUIT - signals that the user has closed *dBASE Plus* by

Clicking the X in the upper right of the frame window

Selecting "Close" from the menu which appears when right clicking on the title bar of the frame window.

Pressing "Alt-F4".

allUsersPath

Topic group

Contains the full path to the folder where shared files or folders may be located for dBASE Plus or for a dBASE Plus application.

Property of

_*app* object

Description

allUsersPath is set during startup to a subfolder of the special Window's folder intended to hold files or folders that may be shared by all authenticated users.

An application may install files and folders under this folder which contain data shared by all users of an application - such as shared configuration settings, shared images or other resources, and other shared data files. In addition, master copies of files that will be copied for each user into their private folder tree, can be installed here as well.

How allUsersPath is set

During startup dBASE or the dBASE runtime engine retrieves the current user's shared application folder path from Windows. The Windows designation for this path is: CSIDL_COMMON_APPDATA.

The path retrieved looks like one of the following paths:

On Windows XP,

C:\Documents and Settings\All Users\Application Data

On newer versions of Windows, this folder defaults to:

C:\ProgramData

Next, the subpath is determined based on the folder from which dBASE or an application .exe was launched.

If launched from under \Program Files or \Program Files (x86), the path remaining after dropping this initial folder is appended to the shared application folder path.

For example, when running dBASE Plus from its default location:

On Windows XP, allUsersPath is set to:

C:\Documents and Settings\All Users\Application Data\dBASE\PLUS\BIN

On newer versions of Windows, allUsersPath is set to:

C:\ProgramData\dBASE\PLUS\BIN

If launched from some other folder, the path remaining after dropping the initial drive specifier or UNC path is appended to the shared application folder path.

For example, when running a dBASE application from:

C:\MyApp\myapp.exe

allUsersPath will be set as follows:

On Windows XP: C:\Documents and Settings\All Users\Application Data\MyApp

On newer versions of Windows: C:\ProgramData\MyApp

When running a dBASE application via a UNC path:

\\SomeUNC\MyApp\myapp.exe

allUsersPath will be set as follows:

On Windows XP: C:\Documents and Settings\All Users\Application Data\MyApp

On newer versions of Windows: C:\ProgramData\MyApp

attach( )

Topic group Related topics

Establishes link between a toolbar and a form.

Syntax

<toRef>.*attach*(<oRef>)

<toRef>

An object reference to the toolbar.

<oRef>

An object reference to the form.

Property of

ToolBar

Description

Along with *detach*( ), this method lets you manage toolbars in your application by connecting and disconnecting the objects as needed.

Typically, however, a toolbar is attached when a form calls a program in which the toolbar is defined, as is done in the included CLIPBAR.PRG sample:

```
parameter FormObj, bLarge
private typeCheck
local t, bNew
bNew = false
if ( PCOUNT() == 0 )
MSGBOX("To attach this toolbar to a form use: " + ;
CHR(13) + CHR(13) + ;
"DO " + PROGRAM() + " WITH <form reference>","Alert")
else
typeCheck = FindInstance("ClipToolbar")
if ( TYPE("typeCheck") == "O" )
t = typeCheck
else
SET PROCEDURE TO (PROGRAM()) ADDITIVE
t = new ClipToolbar( bLarge )
bNew := true
endif
t.attach( FormObj )
endif
return ( bNew )
```

charSet

Topic group Related topics

Returns the name of the global character set.

Property of

_app object

Description

Use the *charSet* property to display the name of the current global character set. This is the same name returned by LIST STATUS or DISPLAY STATUS in the Command window. The charSet property is read-only.

checked

Topic group Related topics Example

Determines if a checkmark appears beside a menu item.

Property of

Menu

Description

Use *checked* to indicate that a condition or a process is turned on or off.

The checkmark appears to the left of the menu command when you set the *checked* property to *true*; the checkmark is removed when you set the *checked* property to *false*.

You may specify a bitmap to display instead of the checkmark with the *checkedBitmap* property, and a bitmap to display when *checked* is *false* with the *uncheckedBitmap* property.

checkedBitmap

Topic group Related topics

A bitmap to display instead of a checkmark when a menu item is *checked*.

Property of

Menu

Description

Use *checkedBitmap* to display a bitmap instead of a checkmark when a menu item's *checked* property is *true*.

The *checkedBitmap* setting can take one of two forms:

RESOURCE <resource id> <dll name>
specifies a bitmap resource and the DLL file that holds it.

FILENAME <filename>
specifies a bitmap file. See class Image for a list of bitmap formats supported by *dBASE Plus*.

Note

The display area in the menu item is very small (13 pixels square with Small fonts). If the bitmap is too large, the top left corner is displayed. Also, the color of the bitmap when the menu item is highlighted changes, depending on the system menu highlight color. Therefore, you may want to limit yourself to simple monochrome bitmaps.

CLEAR TYPEAHEAD

Topic group Related topics

Clears the typeahead buffer, where keystrokes are stored while *dBASE Plus* is busy.

Syntax

CLEAR TYPEAHEAD

Description

If you have not issued a SET TYPEAHEAD TO 0 command, the keyboard typeahead buffer stores keystrokes the user enters while *dBASE Plus* is busy processing other data. When the processing is complete and keyboard input is enabled again, *dBASE Plus* processes and deletes the values in the buffer in the order they were entered until the buffer is empty. Use CLEAR TYPEAHEAD to discard any keystrokes that may have been entered during processing, to ensure that the keyboard data currently being processed comes directly from the keyboard.

For example, if you want to be able to fill in multiple screens quickly, one after the other, you would not issue CLEAR TYPEAHEAD during processing. This would let you continue typing data while data from one screen was being saved and the next (blank) one being displayed. The data you entered during processing would be entered onto the new screen when it appeared.

On the other hand, if you want to make sure that no data is entered until the next screen is displayed, issue CLEAR TYPEAHEAD after displaying the blank screen and before beginning data entry.

currentUserPath

Topic group

Contains the full path to the current user's private dBASE folder or private dBASE application folder.

Property of

_*app* object

Description

currentUserPath is set during startup to a subfolder of the current user's private folder tree where the user's private program settings may be kept and any temporary files can be created, used, and deleted.

Many operating systems, including Microsoft Windows, setup a private folder tree for each user in which their personal files, program settings, and temporary files can be kept.

In Windows XP, the private user folders are located under C:\Documents and Settings.

In newer versions of Windows, the private user folders are located under C:\Users

For dBASE Plus and dBASE applications, a user's private program settings are stored in an .ini file.

- When running dBASE Plus, these settings are stored in a file named plus.ini

- When running an application .exe, these settings are stored in an .ini file named after the application .exe file. For myapp.exe, the .ini file would be named myapp.ini.

The default location for these .ini files is the path contained in *currentUserPath*.

Note that property _app.inifile contains the full path to the .ini file including the name of the .ini file.

For example, for a user with Windows username: jsmith

? _app.inifile returns..

C:\Users\jsmith\AppData\Local\dBASE\PLUS\BIN\Plus.ini

How currentUserPath is set :

During startup dBASE or the dBASE runtime engine retrieves the current user's private folder path from Windows. The Windows designation for this path is: CSIDL_LOCAL_APPDATA.

The path retrieved looks something like:

C:\Users\jsmith\AppData\Local\

Next, the subpath is determined based on the folder from which dBASE or an application .exe was launched.

If launched from under \Program Files or \Program Files (x86), the path remaining after dropping this initial folder is appended to the user's private folder path.

For example, for user jsmith, running dBASE Plus from its default location:

C:\Users\jsmith\AppData\Local\dBASE\PLUS\BIN

If launched from some other folder, the path remaining after dropping the initial drive specifier or UNC path is appended to the user's private folder path.

For example, for user jsmith, running a dBASE application from:

C:\MyApp\myapp.exe

currentUserPath will be set to:

C:\Users\jsmith\AppData\Local\MyApp

databases

Topic group Related topics

An array containing references to all database objects used by the Navigator.

Property of

_app object

Description

Use the *databases* property to reference an array of database objects associated with the _app object. The default database, _app.databases[1], is the first element in that array.

To add a database to the array:

```
d = new database()
d.databaseName = "MyBDEAlias"
d.active = true
_app.databases.add(d)
```

To work with tables referenced by that alias:

```
_app.databases[2].copyTable( "Stuff", "CopyOfStuff" )
```

ddeServiceName

Topic group Related topics

The name used to identify each instance of *dBASE Plus* when used as a DDE (Dynamic Data Exchange) service

Property of
_app object

Description

The _app object's *ddeServiceName* property contains the service name for the current instance of *dBASE Plus*; the default is "DBASE". You may change the *ddeServiceName* if there is more than one instance of *dBASE Plus* running, or if you want to identify your *dBASE Plus* application with a specific DDE service name.

DEFINE COLOR

Topic group Related topics Example

Creates and names a customized color.

Syntax

DEFINE COLOR <color name>
<red expN>, <green expN>, <blue expN>

<red expN>, <green expN>, <blue expN>

Specifies the proportions of red, green, and blue (RGB) that make up the defined color. Each number determines the intensity of the color it represents, and can range from 0 (least intensity) to 255 (greatest intensity).

Description

Use DEFINE COLOR to create a custom color. Once you have defined <color name>, you can use it instead of one of the standard colors such as R, W, BG, silver, lemonchiffon, and so on.

The color you create with DEFINE COLOR is based on three numbers, <red expN>, <green expN>, and<blue expN>. Adjusting these numbers alters the color you create. For example, increasing or decreasing <green expN> increases or decreases the amount of green contained in the customized color.

Use the GETCOLOR( ) function to open a dialog box in which you create a custom color or choose from a palette of available colors. After exiting GETCOLOR( ), issue DEFINE COLOR with the values it returns to define the desired color.

You can't override any standard color definitions. For a full list of standard colors, see the *colorNormal* property.

Colors defined with DEFINE COLOR are active only during the current *dBASE Plus* session. If you restart *dBASE Plus*, you must redefine the colors. You may redefine a custom color as often as you wish. Changing the definition of a color does not automatically change the color of objects that have been set to that color; you must reassign the color.

detach( )

Topic group Related topics

Breaks links between a toolbar and a form.

Syntax

<toRef>.*detach*(<oRef>)

<toRef>

An object reference to the toolbar.

<oRef>

An object reference to the form.

Property of

ToolBar

Description

Along with *attach*( ), this method lets you manage toolbars in your application by connecting and disconnecting the objects as needed.

Typically, however, a toolbar is detached as part of a form's cleanup routines, as is done in the following example:

```
function close
```

```
private sFolder
sFolder = this.restoreSet.folder
CLOSE FORMS
SET DIRECTORY TO &sFolder.
this.toolbars.appbar.detach( _app.framewin )
with (_app)
framewin.text := this.restoreSet.frameText
speedbar := this.restoreSet.speedBar
app := null
endwith
shell( true, true )
return
```

detailNavigationOverride

Topic group Related topics

Controls whether or not a rowset's *navigateMaster* and *navigateByMaster* properties are overridden.

Property of

_app object

Description

Value	Description
0	Use rowset's detail settings
1	Always Navigate Detail Rowsets
2	Never Navigate Detail Rowsets

0 - Use rowset's actual *navigateMaster* and *navigateByMaster* properties to determine whether or not to navigate through detail rowsets when navigating through a master rowset.

1 - All rowsets display SET SKIP behavior. Rowsets behave as if their *navigateMaster* and *navigateByMaster* properties were set to *true*.

2 - All rowsets will not display SET SKIP behavior. Rowsets behave as if their *navigateMaster* and *navigateByMaster* properties were set to *false*.

The default setting for *detailNavigationOverride* is 0.

Whenever you change the value of *detailNavigationOverride*, calling the *refresh*( ) method of any grid with datalinked rowsets will cause the grid to display correctly.

editCopyMenu

Topic group Related topics

Specifies a menu item that copies selected text from a control to the Windows Clipboard.

Property of

MenuBar

Description

The *editCopyMenu* property contains a reference to a menu object users select when they want to copy text.

You can use the *editCopyMenu* property of a form's menuBar to copy selected text to the Windows Clipboard from any edit control in the form, instead of using the control's *copy*( ) method. In effect, *editCopyMenu* calls *copy*( ) for the active control. This lets you provide a way to copy text with less programming than would otherwise be needed. The Copy menu item is automatically disabled when no text is selected, and enabled when text is selected.

For example, suppose you have a Browse object (b) and an Editor object (e) on a form (f). To implement text copying, you could specify actions that would call b.*copy*( ) or e.*copy*( ) whenever a user wanted to copy text. However, if you use a menuBar, you can set the *editCopyMenu* property to the menu item the user will select to copy text. Then, when the user selects that menu item, the text is automatically copied to the Windows Clipboard from the currently active control. You don't need to use the control's *copy*( ) method at all.

If you use the Menu designer to create a menubar, *editCopyMenu* is automatically set to an item named Copy on a pulldown menu named Edit when you add the Edit menu to the menubar:

```
this.EditCopyMenu = this.Edit.Copy
```

editCutMenu

Topic group Related topics

Specifies a menu item that cuts selected text from a control and places it on the Windows Clipboard.

Property of

MenuBar

Description

The *editCutMenu* property contains a reference to a menu object users select when they want to cut text.

You can use the *editCutMenu* property of a form's menubar to cut (delete) selected text and place it on the Windows Clipboard from any edit control in the form, instead of using the control's *cut*( ) method. In effect, *editCutMenu* calls *cut*( ) for the active control. This lets you provide a way to copy text with less programming than would otherwise be needed. The Cut menu item is automatically disabled when no text is selected and enabled when text is selected.

For more information, see *editCopyMenu*.

editPasteMenu

Topic group Related topics

Specifies a menu item that copies text from the Windows clipboard to the currently active edit control.

Property of

MenuBar

Description

The *editPasteMenu* property contains a reference to a menu object users select when they want to paste text to the cursor position in the currently active edit control.

You can use the *editPasteMenu* property of a form's menubar to paste text from the Windows Clipboard into any edit control in the form, instead of using the control's *paste*( ) method. In effect, *editPasteMenu* calls *paste*( ) for the active control. This lets you provide a way to paste text with less programming than would otherwise be needed. The Paste menu item is automatically disabled when the clipboard is empty, and enabled when text is copied or cut to the Clipboard.

For more information, see *editCopyMenu*.

editUndoMenu

Topic group Related topics

Specifies a menu item that reverses the effects of the last Cut, Copy, or Paste action.

Property of

MenuBar

Description

The *editUndoMenu* property contains a reference to a menu object users select when they want to undo their last Cut, Copy, or Paste action.

You can use the *editUndoMenu* property of a form's menubar to undo a Cut or Paste action from any edit control in the form, instead of using the control's *undo*( ) method. In effect, *editUndoMenu* calls *undo*( ) for the active control. This lets you provide a way to undo with less programming than would otherwise be needed.

For more information, see *editCopyMenu*.

errorAction

Topic group Related topics

Default action to be taken when an error is encountered.

Property of

_app object

Description

Use the *errorAction* property to handle errors, and error reporting, when a TRY. . . CATCH or ON ERROR handler is not in affect. If the error occurs within a try...catch, or if an on error handler is active, the TRY...CATCH or ON ERROR handlers will retain control.

errorAction is an enumerated property with the following values:

Value	Description
0	Quit
1	Send HTML Error & Quit
2	Log Error & Quit
3	Send HTML Error, Log Error & Quit
4	Show Error Dialog (Default)
5	Log Error & Show Error Dialog

Option 0

dBASE Plus will shutdown cleanly without reporting the error.

Options 1 and 3

dBASE Plus will to use the file specified in *errorHtmFile* as a template to format error information sent out as an HTML page. These options are only appropriate when running a *dBASE Plus* Web application that was invoked via a web server such as Apache or IIS (Internet Information Server).

The default value for *errorHtmFile* is error.htm.

If error.htm cannot be found, or an error occurs when trying to open or read it into memory, *dBASE Plus* will use a built-in default HTML template that matches the default US English error.htm.

Options 2, 3, and 5

dBASE Plus will log the error to disk using the file specified in *errorLogFile*. If the error log file cannot be opened, for instance when another application has it opened exclusively, *dBASE Plus* will try up to 15 times to open the file and write the log entry. *dBASE Plus* will wait approximately 300 milliseconds between retries. Should all 15 attempts fail, *dBASE Plus* will proceed without logging the error.

Option 4

This is the default option which displays an error dialog offering the option to Fix, Ignore, or Cancel the error.

Option 5

dBASE Plus will log the error and display an error dialog. The error dialog offers the option to Fix, Ignore, or Cancel the error.

errorHTMFile

Topic group Related topics

Filename of an HTM file template. Used for runtime web apps only.

Property of

_app object

Description

Use the *errorHTMFile* property to designate a name for the HTM file used to format an error page sent back to the browser. The filename may include an explicit path or source alias. When a path is not included, the application .exe path is assumed. The default filename for *errorHTMFile* is Error.HTM. *dBASE Plus*'s error handling code will only try to use the *errorHtmFile* property when *errorAction* is set to option 1 - Send HTML Error & Quit or 3 - Send HTML Error, Log Error & Quit.

HTM file template may contain the following replaceable tokens:

Token	Description
%d	Date and Time of error
%e	Application EXE filename

%s	Source filename
%p	Name of procedure or function in which error occurred
%l	Source line number
%c	Location where error code is displayed
%m	Location where error message is displayed

errorLogFile

Topic group Related topics

Specifies the filename of the error log file to be used when the _app objects' *errorAction* property is set to 2, 3, or 5.

Property of

_app object

Description

Use the *errorLogFile* property to designate a filename for the error log file generated when an error occurs, and the *errorAction* property is set to option 2, 3 or 5. The filename may include an explicit path. When a path is not included, the path is set as follows:

- When _app.useUACPaths is True:

  the path returned by _app.currentUserPath is used

- When _app.useUACPaths is False:

  the path returned by _app.exeName is used

The default filename for *errorLogFile* is PLUSErr.log

Information is saved to the log file in the following order:

Date and Time of error

Application Path and Filename (as contained in _app.exeName)

When running PLUS.exe, this will be the path to PLUS.exe.

When running an application exe, this will be the full path and name of the application exe.

Source File Name (if available)

Procedure or Function Name (if available)

Line Number (if available)

Error Code

Error Message

errorLogMaxSize

Topic group Related topics

Approximate maximum size of error log file in kilobytes.

Property of

_app object

Description

The *errorLogMaxSize* property is used to limit the size of an error log file. On a web server or regular file server, limiting the error log size prevents slowly using up all available disk space.

When the size of the file specified in *errorLogFile* exceeds *errorLogMaxSize* x 1024, *dBASE Plus* will skip past the first 10 percent of log entries, find the start of the next complete log entry, and copy the remaining 90 percent of the log file to a new file. Once the log file has been copied successfully, the original log file is deleted and the new log file is renamed to the name specified in *errorLogFile*.

If you do not want to limit the size of the error log file, set *errorLogMaxSize* to zero (0).

The default for *errorLogMaxSize* is 100. The minimum value allowed is zero and the maximum is 2048.

errorTrapFilter

Topic group

Use the *errorTrapFilter* property to enable, or disable, the detection of certain kinds of errors.

Property of

_app object:

Default

0 (Trap all errors)

Description

This property can be set:

Programmatically, by assigning the desired value from a dBL program.

or

Via a setting in the *dBASE Plus* or application ini file as follows:

```
[ErrorHandling]
ErrorTrapFilter=0
```

or

```
[ErrorHandling]
ErrorTrapFilter=1
```

Currently supported options are:

0 Trap all errors. Provides the same level of error trapping introduced in *dBASE Plus* 2.5

1 Ignore interpreter memory access violations. Provides the same level of error trapping available in versions of *dBASE Plus* prior to 2.5.

executeMessages( )

Topic group

Use the *executeMessages*( ) method to periodically process pending messages while running a lengthy processing routine.

Syntax

<oRef>.*executeMessages*( )

<oRef>

A reference to the _app object

Property of

_app object

Description

When called, *executeMessages*( ) checks for messages in the dBASE message queue. If found, they are processed and executed.

While the dBASE interpreter is executing, messages may accumulate in the dBASE message queue - typically mouse, keyboard or paint messages. By calling the *executeMessages*( ) method during a long processing routine, dBASE can be made responsive to these messages rather than having to wait until the processing routine ends.

exeName

Topic group

The drive, path and filename of the currently running instance of PLUS.exe or a *dBASE Plus* application .exe.

Property of

_app object

Description

exeName is a read-only property used to support error handling.

When running PLUS.exe, the *exeName* property will include the drive, path, and file name for the currently running instance of PLUS.exe.

For example:

```
C:\Program Files\dBASE\Plus\bin\PLUS.exe
```

When running a *dBASE Plus* application .exe, the *exeName* property will include the drive, path, and file name of the running application.

In both instances, the *exeName* property preserves the case of the folder and .exe names (except on Win 9x where it is converted to uppercase). When starting an application .exe as a parameter to PLUSrun.exe, the *exeName* property will use the case entered on the command line for the applications path & name.

GETCOLOR( )

Topic group Related topics Example

Calls a dialog box in which you can define a custom color or select a color from the color palette. Returns a character string containing the red, green, and blue values for the color selected.

Syntax

GETCOLOR([<title expC>])

<title expC>

A character string to appear as the title of the dialog box.

Description

Use GETCOLOR( ) to open a dialog box in which you can choose a color from a palette of predefined colors or create a customized color. In this dialog box, you choose and create colors in the same way you do if you use the Color Palette available when you choose Color in the Windows Control Panel.

GETCOLOR( ) returns a string in the format "red value, green value, blue value", with each color value ranging from 0 to 255; for example "115,180,40". If you cancel the color dialog, GETCOLOR( ) returns an empty string.

You can use the string returned by GETCOLOR( ) in a related command, DEFINE COLOR, to use a specific color in a program.

GETFONT( )

Topic group Example

Calls a dialog box in which you select a character font. Returns a string containing the font name, point size, font style (if you choose a style other than Regular), and family.

Syntax

GETFONT([<title expC>
[, <fontstr expC>]])

<title expC>

A character string to appear as the title of the dialog box. Whenever the <fontstr expC> parameter is used, the <title expC> parameter must also be present, as a valid title string or a null string (""), in order to specify the use of the default dialog title.

<fontstr expC>

A character string containing the default font settings to be used in the dialog box. This string has the same format as the results string returned by this dialog,

```
fontstr = “fontName, pointsize [, [B] [I] [U] [S] ] [,fontFamily]”
```

where the style options, B => Bold, I => Italic, U => Underline, and S => Strikeout, can appear in any order, and in either upper or lower case.

The following are valid examples of the GETFONT( ) syntax:

```
GETFONT()
GETFONT("My Title")
GETFONT("","Arial,14,BU")
```

Whereas,

```
GETFONT( ,"Arial,14,BU")
```

will result in an error dialog.

Description

Use GETFONT( ) to place the values associated with a specified font into a character string. If you want to add a font to the [Fonts] section of PLUS.ini but don't know its exact name or family, use GETFONT( ). Then add the information GETFONT( ) returns into PLUS.ini.

hasHScrollBar( )

Topic group Related topics Example

Use the *hasHScrollBar*( ) method to determine if a frame window is using a horizontal scrollbar.

Syntax

<oRef>.*hasHScrollBar*( )

<oRef>

<oRef> a reference to the _app.FrameWin object.

Property of

_app.frameWin

Description

By indicating whether a horizontal scrollbar is present, the *hasHScrollBar*( ) method allows a form to more accurately determine how much room is available within the frame window.

The *hasHScrollBar*( ) method returns True if the frame window has a horizontal scrollbar.

hasVScrollBar( )

Topic group Related topics Example

Use the *hasVScrollBar*( ) method to determine if a frame window is using a vertical scrollbar.

Syntax

<oRef>.*hasVScrollBar*( )

<oRef>

<oRef> a reference to the _app.FrameWin object.

Property of

_app.frameWin

Description

By indicating whether a vertical scrollbar is present, the *hasVScrollBar*( ) method allows a form to more accurately determine how much room is available within the frame window.

The *hasVScrollBar*( ) method returns True if the frame window has a vertical scrollbar.

INKEY( )

Topic group Related topics Example

Gets the first keystroke waiting in the keyboard typeahead buffer. Can also be used to wait for a keystroke and return its value.

Syntax

INKEY([<seconds expN>] [, <mouse expC>])

<seconds expN>

The number of seconds INKEY( ) waits for a keystroke. Fractional times are allowed. If <expN> is zero, INKEY( ) waits indefinitely for a keystroke. If <expN> is less than zero, the parameter is ignored.

<mouse expC>

Determines whether INKEY( ) returns a value when you click the mouse. If <expC> is "M" or "m", INKEY( ) returns –100. If <expC> is not "M" or "m", INKEY( ) ignores a mouse click and waits for a keystroke.

Description

The keyboard typeahead buffer stores keystrokes the user enters while *dBASE Plus* is busy. A very fast typist may also fill the keyboard typeahead buffer—*dBASE Plus* is busy trying to keep up. These keystrokes are normally handled automatically; for example, characters are typed into entryfields and menu choices are made. Use INKEY( ) to handle the keystrokes yourself.

INKEY( ) returns the decimal value associated with the first key or key combination held in the keyboard typeahead buffer and removes that keystroke from the buffer. If the typeahead buffer is empty, INKEY( ) returns the value of zero.

Key pressed	Return value	Shift+key return value	Ctrl+Key return value	Alt+key* return value
0	48	Depends on keyboard	–404	–452
1	49	Depends on keyboard	–404	–451
2	50	Depends on keyboard	–404	–450
3	51	Depends on keyboard	–404	–449
4	52	Depends on keyboard	–404	–448
5	53	Depends on keyboard	0	–447
6	54	Depends on keyboard	–30	–446
7	55	Depends on keyboard	–404	–445
8	56	Depends on keyboard	–404	–444
9	57	Depends on keyboard	–404	–443
a	97	65	1	–435
b	98	66	2	–434
c	99	67	3	–433
d	100	68	4	–432
e	101	69	5	–431
f	102	70	6	–430
g	103	71	7	–429
h	104	72	8	–428
i	105	73	9	–427
j	106	74	10	–426
k	107	75	11	–425
l	108	76	12	–424
m	109	77	13	–423
n	110	78	14	–422
o	111	79	15	–421
p	112	80	16	–420
q	113	81	17	–419
r	114	82	18	–418
s	115	83	19	–417
t	116	84	20	–416
u	117	85	21	–415
v	118	86	22	–414

w	119	87	23	–413
x	120	88	24	–412
y	121	89	25	–411
z	122	90	26	–410
F1 (Ctrl+\)	28	–20	–10	–30
F2	–1	–21	–11	–31
F3	–2	–22	–12	–32
F4	–3	–23	–13	–33
F5	–4	–24	–14	–34
F6	–5	–25	–15	–35
F7	–6	–26	–16	–36
F8	–7	–27	–17	–37
F9	–8	–28	–18	–38
F10	–9	–29	–19	–39
F11	–544	–546	–548	–550
F12	–545	–547	–549	–551
Left Arrow	19	–500	1	0
Right Arrow	4	–501	6	0
Up Arrow	5	5	5	0
Down Arrow	24	24	24	0
Home (Ctrl+])	26	26	29	0
End	2	2	23	0
Tab	9	–400	0	0
Enter	13	0	–402	0
Esc (Ctrl+[)	27	27	-	-
Ins	22	0	0	0
Del	7	–502	7	7
Backspace	127	127	–401	–403
PgUp	18	18	31	0
PgDn	3	3	30	0

Note

The Alt+key value returned for all character keys, except lower-case letters a through z, is the character value minus 500. For lower-case letters, the Alt+key values are the same as those for upper-case letters.

Because of the event-driven nature of *dBASE Plus*, INKEY( ) is rarely used. When it is used, it's in one of three ways:

- When keystrokes are expected to be buffered, INKEY( ) is used to get those keystrokes.
- In a loop that's busy doing something, INKEY( ) is used to see if a key has been pressed, and if so to take an action.
- INKEY( ) can be used to wait for a keystroke, and then take an action.

In any of these cases, because *dBASE Plus* is busy executing your INKEY( ) code, it will not respond to keystrokes and mouse clicks as it normally would.

To check if there is a key waiting in the buffer without removing it, or to determine a value in the buffer in a position other than the first position, use NEXTKEY( ).

KEYBOARD

Topic group Related topics Example

Inserts keystrokes into the typeahead buffer.

Syntax

KEYBOARD <expC> [CLEAR]

<expC>

A character string, which may include mnemonic strings representing key labels.

CLEAR

Empties the typeahead buffer before inserting <expC>.

Description

Use KEYBOARD to simulate keystrokes by placing or "stuffing" them in the typeahead buffer.

KEYBOARD can place any number of characters in the typeahead buffer, up to the limit specified by SET TYPEAHEAD; subsequent characters are ignored. If SET TYPEAHEAD is 0, you may KEYBOARD one character.

Keystrokes simulated with KEYBOARD are treated like normal keystrokes, going into the control that currently has focus. Some controls support a *keyboard*( ) method that enables you to send keystrokes to that specific control.

In addition to the simple alphanumeric keys on the keyboard, you may also use mnemonic strings to simulate must function keys. For a list of mnemonic strings, see the *keyboard*( ) method.

language [_app object]

Topic group

Identifies the display language currently used in the design and runtime environments. This property is read only.

Property of

_app object

Description

Use the app object's *language* property to determine which language version of *dBASE Plus* was installed. It's value is read on startup from the Plus.ini file or, in the case of deployed *dBASE Plus* applications, from the applications .INI file.

If multiple languages have been installed, the value of the *language* property is determined by the first language installed (that which was written to the PLUS.ini file), or a language selected via the Desktop Properties dialog. Languages selected via the Desktop Properties dialog will only allow a change if you have the appropriate language .DLLs installed.

lDriver

Topic group Related topics

Returns the global language driver name and description.

Property of

_app object

Description

Use the lDriver property to view information about the current global language driver. Information displayed, name and description, is the same as that returned by LIST STATUS or DISPLAY STATUS in the Command window. The lDriver property is read-only.

MSGBOX( )

Topic group Related topics Example

Opens a dialog box that displays a message and pushbuttons, and returns a numeric value that corresponds to the pushbutton the user chooses.

Syntax

MSGBOX(<message expC>, [<title expC>, [<box type expN>]])

<message expC>

The message to display in the dialog box.

<title expC>

The title to display in the title bar of the dialog box.

<box type expN>

A numeric value that determines which icon (if any) and which pushbuttons to display in the dialog box. To specify a dialog box with pushbuttons and no icon, use the following numbers:

<box type expN>	Pushbuttons
0	OK
1	OK, Cancel
2	Abort, Retry, Ignore
3	Yes, No, Cancel
4	Yes, No
5	Retry, Cancel

To specify a dialog box with pushbuttons and an icon, add any of the following numbers to <box type expN>:

Number to add	Icon displayed
16	
32	
48	

64

When a dialog box has more than one pushbutton, the left most pushbutton is normally the default, However, if you add 256 to <*box type expN*>, the second pushbutton is the default, and if you add 512 to <*box type expN*>, the third pushbutton is the default.

If you omit <box type expN>, box type 0—one with just the title, message, and an OK button—is used by default.

Note

If you specify <box type expN>, make sure it's a valid combination of the choices outlined above. An invalid number may result in a dialog box that you cannot close.

Description

Use MSGBOX( ) to prompt the user to make a choice or acknowledge a message by clicking a pushbutton in a modal dialog box.

While the dialog box is open, program execution stops and the user cannot give focus to another window. When the user chooses a pushbutton, the dialog box disappears, program execution resumes, and MSGBOX( ) returns a numeric value that indicates which pushbutton was chosen.

Pushbutton	Return value
OK	1
Cancel	2
Abort	3
Retry	4
Ignore	5
Yes	6
No	7

NEXTKEY( )

Topic group Related topics Example

Checks for and returns a keystroke held in the keyboard typeahead buffer.

Syntax

NEXTKEY([<expN>])

<expN>

The position of the key or key combination in the typeahead buffer. If <expN> is omitted, NEXTKEY( ) returns the value of the first keystroke in the buffer. If <expN> is larger than the number of keystrokes in the buffer, NEXTKEY( ) returns 0.

Description

The keyboard typeahead buffer stores keystrokes the user enters while *dBASE Plus* is busy. A very fast typist may also fill the keyboard typeahead buffer—*dBASE Plus* is busy trying to keep up. These keystrokes are normally handled automatically; for example, characters are typed into entryfields and menu choices are made. Use NEXTKEY( ) to check if there are any buffered keystrokes, or to look for a keystroke in a specific position in the buffer.

NEXTKEY( ) returns the decimal value associated with the key or key combination held in the keyboard typeahead buffer at the specified position in the buffer. Unlike INKEY( ), NEXTKEY( ) does not remove the keystroke from the buffer. If the buffer is empty or there is no keystroke at the specified position, NEXTKEY( ) returns a value of zero.

For a list of keystroke values, see INKEY( ).

ON ESCAPE

Topic group Related topics

Changes the default behavior of the Esc key so that it executes a specified command instead of interrupting command or program execution.

Syntax

ON ESCAPE [<command>]

<command>

The command to execute when the following conditions are in effect:

SET ESCAPE is ON
The user presses Esc during command or program execution

The <command> may be any valid dBL command, including a DO command to execute a program file, a function call that executes a program or function loaded in memory, or a codeblock.

ON ESCAPE without a <command> option disables any previous ON ESCAPE <command>.

Description

The primary purpose of the Esc key is to interrupt command or program execution. This behavior may be changed with ON ESCAPE; either way it occurs only when SET ESCAPE is ON (its default setting).

When no ON ESCAPE <command> is in effect, pressing Esc interrupts program execution and displays the *dBASE Plus* Program Interrupted dialog box. If ON ESCAPE <command> is in effect, pressing Esc during program execution executes the specified command instead and then continues program execution.

While executing a command (like CALCULATE) from the Command window, pressing Esc with ON ESCAPE <command> in effect executes <command> and then terminates the command, returning control to the Command window. If no ON ESCAPE <command> is in effect, pressing Esc during a command from the Command window simply terminates that command and displays a message in the status bar.

Note that user interface elements such as menus, forms, and dialog boxes handle Esc differently, usually closing or dismissing that UI element. (For forms, this behavior is controlled by its escExit property.) In those cases, ON ESCAPE and SET ESCAPE have no effect. In fact, with the exception of dialog boxes and forms opened with ReadModal( ), because of the event-driven nature of *dBASE Plus* there is no program executing when you use a menu or type into a form, so there is nothing to interrupt.

Use ON KEY to specify a new meaning or mapping for Esc or any other key. If both ON KEY and ON ESCAPE are in effect, ON KEY takes precedence when Esc is pressed. In other words, while ON ESCAPE changes the Escape behavior, ON KEY changes the meaning of the Esc key, so that pressing it no longer causes that Escape behavior. While the Escape behavior affects only programs or commands that are executing, ON KEY works at all times.

If you issue ON ESCAPE<command> in a program, you should disable the current ON ESCAPE condition by issuing ON ESCAPE without a <command> option before the program ends. Otherwise, the ON ESCAPE condition remains in effect for any subsequent commands and programs you issue and run until you exit *dBASE Plus*.

ON KEY

Topic group Related topics

Changes the keyboard mapping to execute a command when a specified key or key combination is pressed.

Syntax

ON KEY [LABEL <key label>] [<command>]

LABEL <key label>

Identifies the key or key combination that, when pressed, causes <command> to execute. Without LABEL <key label>, *dBASE Plus* executes <command> when you press any key. ON KEY LABEL is not case-sensitive.

<command>

The command that is executed when you press the key or key combination. If you omit <command>, the command previously assigned by ON KEY is disabled.

Description

Each key on the keyboard has a default meaning or mapping. For alphanumeric keys, this mapping simply types the character. Function keys have predefined actions. The Esc key terminates program execution. Use ON KEY to specify a command that executes when the user presses a key or key combination, overriding the default mapping.

Actions defined by ON KEY will interrupt programs, but not commands; in a program, the ON KEY action will occur after the current command has completed and then the program will continue. ON KEY also doesn't affect the execution of some commands or functions that are specifically looking for keystrokes, such WAIT or INKEY( ). On the other hand, if you KEYBOARD a key that has been remapped, the remapped behavior will occur.

When you issue both ON KEY LABEL <key label> <command> and ON KEY <command>, the key or key combination you specify with ON KEY LABEL <key label> <command> takes precedence and executes its associated <command>. This way you can define actions for specific keys, and a default global action for all other keys. There may be only one ON KEY specification for each specific key and one global action defined at a given time.

ON KEY without arguments removes the effect of all previously-entered ON KEY <command> commands, with or without a LABEL.

To change the default Escape behavior, which interrupts the currently executing program or command, use ON ESCAPE. If both ON KEY and ON ESCAPE are in effect, ON KEY takes precedence when Esc is pressed. In other words, while ON ESCAPE changes the Escape behavior, ON KEY changes the meaning of the Esc key, so that pressing it no longer causes that Escape behavior. While the Escape behavior affects only programs or commands that are executing, ON KEY works at all times.

To assign strings to function keys, use SET FUNCTION. If both ON KEY on SET FUNCTION are in effect, ON KEY takes precedence.

ON KEY LABEL

The <key label> for the standard alphanumeric keys is simply the character on that key, for example, A, b, 2, or @. Use the following key label names to assign special keys or key combinations with ON KEY LABEL <key label>.

Key identification	<key label>
Backspace	Backspace
Back Tab	Backtab
Delete	Del
End	End
Home	Home
Insert	Ins
Page Up	PgUp
Page Down	PgDn
Tab	Tab
Left arrow	Leftarrow
Right arrow	Rightarrow
Up arrow	Uparrow
Down arrow	Dnarrow
F1 to F12	F1, F2, F3, ...
Control+<key>	Ctrl-<key> or Ctrl+<key>
Shift+<key>	Shift-<key> or Shift+<key>
Alt+<key>	Alt-<key> or Alt+<key>
Enter	Enter
Escape	Esc
Space bar	Spacebar

onInitiate

Topic group Related topics Example

Event fired when a client application requests a DDE link with *dBASE Plus* as the server, and no DDETopic object for the specified topic exists in memory.

Parameters

<topic expC>

The requested DDE topic.

Property of

_app object

Description

The *onInitiate* event executes a DDE (Dynamic Data Exchange) initiation-handler routine. You write this routine to create DDETopic objects, which handle DDE server events.

A DDE client application initiates a DDE link by specifying the DDE service name and a topic. The *_app* object's *ddeServiceName* property contains the service name for the current instance of *dBASE Plus*; the default is "DBASE". You may change the *ddeServiceName* if there is more

than one instance of *dBASE Plus* running, or if you want to identify your *dBASE Plus* application with a specific DDE service name.

Once the client application locates the desired *dBASE Plus* server by service name, it attempts to create a link on a specific topic. If a DDETopic object already exists in memory for the named topic, that object is used and the link is completed. If there is no DDETopic object for that topic, the *onInitiate* event fires. The *onInitiate* event handler must then create the DDETopic object, using that topic as a parameter to the constructor, and RETURN the resulting object to complete the link.

onInitMenu

Topic group Related topics

Specifies code that executes when a menubar or popup is opened.

Property of

MenuBar, Popup

Description

The *onInitMenu* event is called whenever a menubar or popup is invoked, and is processed before the menubar's child menus or the popup is displayed.

You can use the *onInitMenu* event to determine the status of menu items that will be displayed. For example, use the *onInitMenu* event to determine if the enabled or checked property of a menu item should be true or false.

onUpdate

Topic group

Fires repeatedly while application is idle to refresh toolbuttons.

Property of

ToolBar

Description

This event maintains the status of toolbuttons on a toolbar by firing whenever the application hosting the toolbar is idle.

roamingUsersPath

Topic group

Contains the full path to the current user's roaming dBASE folder or roaming dBASE application folder

Property of

_*app* object

Description

roamingUsersPath is set during startup to a subfolder of the current user's roaming user profile folder tree where the user's roaming program settings may be kept.

When connected to a Windows network where roaming user profiles are enabled, you can set up a roaming user profile in Microsoft Windows for each user in which their personal files, program settings, and other files can be kept and updated on a server.

When logging into a Windows server via a local workstation, the user's roaming profile files will be copied from the server to the local workstation.

When logging out, the user's roaming profile files are copied back up to the server.

In Windows XP, the roaming user profile folders are located under:

C:\Documents and Settings\<username>\application data\.

In newer versions of Windows, roaming user profile folders are located under:

C:\Users\<username>\AppData\Roaming\

For dBASE Plus and dBASE applications, a user's private program settings are stored in an .ini file.

- When running dBASE Plus, these settings are stored in a file named plus.ini
- When running an application .exe, these settings are stored in an .ini file named after the application .exe file. For myapp.exe, the .ini file would be named myapp.ini.

How to enable a dBASE application to use roamingUsersPath:

When building an application specify: INI ROAM in the BUILD command.

For Example:

BUILD myapp.pro to myapp.exe INI ROAM

This will enable the application to automatically use the path in _app.roamingUsersPath to set the location for its .ini file. This path will be copied into _app.inifile during application startup.

In the dBASE Project Explorer, this can be set by choosing the "Roaming" option for the INI Type? setting on the Project tab.

Caution - in order for roaming profiles to work correctly, any paths stored in an application's .ini file must be valid on any workstation the user chooses to work from.

How roamingUsersPath is set :

During startup dBASE or the dBASE runtime engine retrieves the current user's roaming user profile folder path from Windows. The Windows designation for this path is: CSIDL_APPDATA.

The path retrieved looks something like:

C:\Users\jsmith\AppData\Roaming

Next, the subpath is determined based on the folder from which dBASE or an application .exe was launched.

If launched from under \Program Files or \Program Files (x86), the path remaining after dropping this initial folder is appended to the user's roaming profile folder path.

For example, for user jsmith, running dBASE Plus from its default location:

C:\Users\jsmith\AppData\Roaming\dBASE\PLUS\BIN

If launched from some other folder, the path remaining after dropping the initial drive specifier or UNC path is appended to the user's roaming profile folder path.

For example, for user jsmith, running a dBASE application from:

C:\MyApp\myapp.exe

roamingUsersPath will be set to:

C:\Users\jsmith\AppData\Roaming\MyApp

separator

Topic group

Determines if a menu item is a separator line instead of a menu option.

Property of

Menu

Description

Set the *separator* property to *true* when you want to use a menu item as a separator between groups of menu commands. When the *separator* property is *true*, other properties such *text* and *onClick* are ignored.

SET CONFIRM

Topic group

Controls the cursor's movement from one entry field to the next during data entry.

Syntax

SET CONFIRM ON | off

Description

When SET CONFIRM is OFF, *dBASE Plus* moves the cursor immediately to the next input area when the current one is full. When SET CONFIRM is ON, the cursor moves to the next input area only when you press Enter or a cursor-control key, or when you click another input area with the mouse.

Use SET CONFIRM ON to prevent moving the cursor from one input area to the next automatically, thus avoiding data-entry errors such as the overflow of contents from one input area into the next. Use SET CONFIRM OFF when input speed is more important.

SET CUAENTER

Topic group Related topics

Determines whether Enter simulates Tab in a form.

Syntax

SET CUAENTER ON | off

Default

The default for SET CUAENTER is ON. To change the default, update the CUAENTER setting in PLUS.ini. To do so, either use the SET command to specify the setting interactively, or enter the CUAENTER parameter directly in PLUS.ini.

Description

The CUA (Common User Access) interface standard dictates that the Tab key moves focus from one control to another, while the Enter key submits the entire form (which fires the form's onSelection event). Use SET CUAENTER to control whether Enter follows the CUA standard. If SET CUAENTER is OFF, the Enter key emulates the Tab key, moving the focus to the next control. This behavior is important to note in the case of dialog boxes, where either the OK or Cancel button is specified as the default (See the PushButton's *default* property). When CUAENTER is set to OFF, pressing the Enter key will cause focus to toggle between these buttons, thus preventing the form from being submitted.

There are two good reasons to ignore the CUA behavior:

In many forms, especially ones that take advantage of the numeric keypad, using the Enter key to move to the next control speeds data entry.

For non-trivial forms, there are usually a number of pushbuttons which take completely different actions; for example, adding a new record, deleting the current record, navigating forward, navigating backward, etc. If you press Enter while the focus is in an entryfield for example, the act of submitting the form tells you nothing about what the user wants to do next.

In fact, few applications consider the action of submitting the form; the onSelection event is rarely used. When the *onSelection* event is left undefined, nothing happens when you press Enter (except in a control like the Editor). So you might as well make the Enter key do something useful and have it move the focus to the next control.

Regardless of the setting of SET CUAENTER, you can move focus from object to object with the mouse or by pressing Tab and Shift-Tab.

SET ESCAPE

Topic group

Specifies whether pressing Esc interrupts program execution.

Syntax

SET ESCAPE ON | off

Default

The default for SET ESCAPE is ON. To change the default, set the ESCAPE parameter in PLUS.ini.

Description

The primary purpose of the Esc key is to interrupt command or program execution. While this behavior may be changed with ON ESCAPE, this behavior occurs only when SET ESCAPE is ON.

Typically, SET ESCAPE is ON during application development. This allows you to stop processes which are taking too long or have run amok. When an application is deployed, you should either:

1. SET ESCAPE OFF so that the user cannot cause your application to terminate abnormally, or
   - Define a specific ON ESCAPE behavior so that your application or process can shutdown or be canceled gracefully, usually after confirming that the user really wants to do so.

Note

If SET ESCAPE is OFF and you have not used ON KEY or some other method to interrupt your program, you can interrupt program execution only by forcing the termination of *dBASE Plus* or your *dBASE Plus* application. Forced termination can cause data loss.

Note that user interface elements such as menus, forms, and dialog boxes handle Esc differently, usually closing or dismissing that UI element. (For forms, this behavior is controlled by its *escExit* property). In those cases, ON ESCAPE and SET ESCAPE have no effect. In fact, with the exception of dialog boxes and forms opened with the *readModal*( ) method, because of the event-driven nature of *dBASE Plus* there is no program executing when you use a menu or type into a form, so there is nothing to interrupt.

SET FUNCTION

Topic group Related topics Example

Assigns a string to a function key or to a combination of the Ctrl (control) key or the Shift key and a function key.

Syntax

SET FUNCTION <key> TO <expC>

<key>

A function key number, function key name, or character expression of a function key name—for example, 3, F3, or "F3". Specify a character expression for <key> to assign a key combination using the Ctrl or Shift key with a function key. Type "CTRL+" or "SHIFT+" and then a function key name—for example, "shift+F5" or "Ctrl+f3". The function key names are not case-sensitive and you may use a hyphen in place of the plus sign. You can't combine Ctrl and Shift, such as "Ctrl+Shift+F3".

<expC>

Any character string, often the text of a command. Use a semicolon (;) to represent the Enter key. Placing a semicolon at the end of a command has the effect of executing that command when you press the function key in the Command window. You can execute more than one command by separating each command in the list with a semicolon.

Default

The following function key settings are in effect when *dBASE Plus* starts:

Key	Command	Key	Command
F1	HELP;	F7	DISPLAY MEMORY;
F3	LIST;	F8	DISPLAY;
F4	DIR;	F9	APPEND;
F5	DISPLAY STRUCTURE;	F10	Activates the menu
F6	DISPLAY STATUS;		

Description

Use SET FUNCTION to simulate typing a string with a single keystroke. These strings are usually commands to be executed in the Command window, or common strings used in data entry.

Note

F2 is reserved for toggling between views while in the Browse window. You can program it, but it will not be recognized when in the Browse window. You cannot program F10, or any combination using F11 or F12. You cannot program keys that are used as standard Windows functions, such as Ctrl-F4.

When you press the programmed function key or key combination, the assigned string appears at the cursor. Strings for the Command window usually end in a semicolon, which represents the Enter key. The simulated Enter key causes the command to be executed immediately.

While SET FUNCTION is specifically intended to simulate typing a string, you can use the ON KEY command to program a function key or any other key to execute any command. For example, these two commands (executed separately, not consecutively):

```
set function f7 to "display memory;"
on key label f7 display memory
```

would both cause the F7 key to execute the DISPLAY MEMORY command if the key was pressed on a blank line in the Command window. But suppose the line in the Command window contained the word "field" and the cursor was at the beginning of that line. Then with SET FUNCTION F7, pressing the function key would cause the string "display memory" to be typed into the line, resulting in "display memoryfield", and then the Enter key would be simulated, causing *dBASE Plus* to attempt to display a field named "memoryfield" in the current workarea. With ON KEY LABEL F7, the DISPLAY MEMORY command would be executed with nothing being typed into the Command window.

If the cursor was in an entryfield for a city in a form, then with SET FUNCTION F7, you would get the city of "display memory" and the cursor would move to the next control if SET CUAENTER was OFF. Again, with ON KEY LABEL F7, the DISPLAY MEMORY command would be executed without affecting the data entry.

To see the list of strings currently assigned to function keys, use DISPLAY STATUS.

SET MESSAGE

Topic group

Specifies the default message to display in the status bar.

Syntax

SET MESSAGE TO [<message expC>]

<message expC>

The message to display

Description

Use SET MESSAGE to set the default message that appears the status bar. Menu items and controls on forms have a statusMessage property. When that object has focus, and that property is not empty, that message is displayed instead.

SET MESSAGE TO without the option <message expC> sets the default message to an empty string, and removes any message from the status bar.

The status bar may be suppressed by setting the _app.statusBar property to *false*.

SET TYPEAHEAD

Topic group Related topics

Sets the size of the typeahead buffer, where keystrokes are stored while *dBASE Plus* is busy.

Syntax

SET TYPEAHEAD TO <expN>

<expN>

the size of the keyboard typeahead buffer, any number from 0 to 1600.

Default

The default size of the typeahead buffer is 50 characters. To change the default, set the TYPEAHEAD parameter in PLUS.ini.

Description

The keyboard typeahead buffer stores keystrokes the user enters while *dBASE Plus* is busy, for example while reindexing a table. When the processing is complete and the application is ready to accept keystrokes, *dBASE Plus* fetches and deletes the values in the buffer in the order they were entered. Any keys typed while there are still keystrokes in the buffer are added to the end of the buffer.

If the size of the typeahead buffer is set to 50, *dBASE Plus* can store values for 50 keypresses; further keystrokes are ignored without any warnings. A large typeahead buffer is useful if the user does not want to stop typing when *dBASE Plus* is unavailable for processing direct keyboard input.

For some programs, you may want to disable the typeahead buffer with SET TYPEAHEAD TO 0. This ensures that user input comes directly from the keyboard, rather than from the typeahead buffer.

For example, if you want to be able to fill in multiple forms quickly, one after the other, you might SET TYPEAHEAD to a relatively high number during form processing. This would let you continue typing data while one form was being saved and the next (blank) one being displayed. The data you entered during processing would be entered onto the new form when it appeared. On the other hand, if you want to make sure that no data is entered until the form is displayed on the screen, you can issue SET TYPEAHEAD TO 0.

You can also clear the typeahead buffer manually with CLEAR TYPEAHEAD.

SET TYPEAHEAD limits the number of characters you can put into the typeahead buffer using KEYBOARD.

SHELL( )

Topic group Related topics

Hides or restores the components of the application shell: the Command window (and Navigator) and the MDI frame window. Returns a logical value corresponding to the previous SHELL( ) state.

Syntax

SHELL([<expL*1*>, [<expL*2*>]])

<expL*1*>

The value that determines whether to display the shell.

<expL2>

The value that determines whether to force the display of the MDI frame window. If <*expL1*> is true, the full shell is on and <expL2> is ignored. If <*expL1*> is false, <expL2> defaults to false.

Description

SHELL( ) controls the display of the components of the application shell:

- The Command window
- The Navigator
- The MDI frame window, which contains the Command window, Navigator, and all MDI forms and their toolbars and menus. This window is represented by the _app.frameWin object.

In *dBASE Plus*, SHELL( ) defaults to true; all three components are displayed. In a compiled application, SHELL( ) defaults to false; none of the elements are displayed, unless either:

- An MDI form is open, in which case the MDI frame window must be displayed to contain the MDI form(s), or
- A menu has been assigned to _app.frameWin.

In either case, the MDI frame window stays visible regardless of the <expL2> value.

shortCut

Topic group Related topics

Specifies a key combination that fires the *onClick* event of a menu object.

Property of

Menu

Description

Use the *shortCut* property to provide a quick way to execute a menu command with the keyboard. For example, if you assign the character string "CTRL+S" to a menu option's *shortCut* property, the user can execute that menu option by pressing Ctrl+S.

The value you specify with the *shortCut* property is displayed next to the prompt you specify with the Text property.

To view a list of dBASE keyboard combinations, see topics:

- dBASE Plus Classic keyboard mappings
- Brief Editor keyboard mappings

SLEEP

Topic group Related topics Example

Pauses a program for a specified interval or until a specified time.

Syntax

```
SLEEP
 <seconds expN> |
 UNTIL <time expC> [,<date expC>]
```

<seconds expN>

The number of seconds to pause the program. The number must be greater than zero and no more than 65,000 (a little over 18 hours). Fractional times are allowed. Counting starts from the time you issue the SLEEP command.

UNTIL <time expC>

Causes program execution to pause until a specified time (<time expC>) on the current day. If you also specify <date expC>, the program pauses until the time on that day. The time and date *dBASE Plus* uses are the system time and date. You can set the system time with SET TIME and the system date with SET DATE TO. If the time has already passed, SLEEP UNTIL <time expC> has no effect.

The <time expC> argument is a 24-hour time that matches the format returned by the TIME( ) function. A typical format for <time expC> is "HH:MM:SS". The delimiter is conventionally a colon but can be changed through the Regional Settings in the Windows Control Panel. The time string must include the seconds.

<date expC>

An optional date until which the program is to pause. The <date expC> argument is a character expression (not a date expression) that represents a date in the current date format; it would match the string returned by the DTOC( ) function. For example, if SET DATE is AMERICAN, the format would be "MM/DD/YY".

If the date has already passed, SLEEP UNTIL <time expC> [,<date expC>] has no effect. If you want to specify a value for <date expC>, you must also specify a value for <time expC>.

Description

Use SLEEP to pause a program either for <seconds expN> seconds or until a specified time (<time expC>). The specified time is the same day the program is running unless you specify a date with <date expC>. If SET ESCAPE is ON, you can interrupt SLEEP by pressing Esc.

Note

If SET ESCAPE is OFF, there is no way to interrupt SLEEP. However, you can use Ctrl+Esc and Alt+Tab to switch to another Windows application, or Alt+F4 to exit *dBASE Plus*.

Although SLEEP can generate a pause from the Command window, programmers use it primarily within programs. For example, you can use SLEEP to generate a pause between multiple displaying windows or to allow a user to read a message on the screen or complete an action. Pauses are also useful when you need to delay program execution until a specific time.

While SLEEP is active, *dBASE Plus* is considered busy; that is, busy sleeping. Program execution is suspended, keystrokes go into the typeahead buffer, and *dBASE Plus* does not respond to events like mouse clicks or timers. If you want an event to occur at a specified time without putting *dBASE Plus* to sleep, use a Timer object.

SLEEP is an alternative to using a DO WHILE loop, a FOR loop, or WAIT to generate pauses in a program. SLEEP is more accurate than using loops because it's independent of the execution speed of the system. You can also use INKEY(<expN>) if you want the user to be able to interrupt the pause and continue with program processing.

sourceAliases

Topic group Related topics

An associative array containing object references to currently defined Source Aliases

Property of

_app object

Description

The *sourceAliases* property is a read-only associative array which contains object references to all source aliases defined in the PLUS.ini. To loop through the elements of the array, use the *firstKey* property:

```
aKey = _app.sourceAliases.firstKey // Get first key in the AssocArray
```

Once you have the key value for the first element, use the *nextKey*( ) method to get key values for the rest of the elements:

```
for nElements = 1 to _app.sourceAliases.count()
 aKey := _app.sourceAliases.nextKey( aKey ) // Get next key value
 ? aKey, _app.SourceAliases[ aKey ]     // display
endfor
```

Values in the sourceAliases array can also be accessed from the Source Aliases section of the Properties | Desktop Properties dialog.

speedBar [_app]

Topic group Related topics

Determines whether to display the default toolbar

Property of

_app object

Description

The value of the _app object's *speedBar* property is determined at *dBASE Plus* startup by the setting stored in PLUS.ini, or an application's .ini file. You can view or change this setting through the Standard option in the PLUS.ini [Toolbars] section.

```
[Toolbars]
Standard=1 // for _app.speedBar = True
   or
Standard=0 // for _app.speedBar = False
```

_app.speedBar defaults to "True".

terminateTimerInterval

Topic group Related topics

Determines the number of milliseconds it takes to remove an orphaned PLUSrun.exe from a web servers memory.

Property of
_app object

Description

The terminateTimerInterval property allows you to set a value for the interval between termination of a *dBASE Plus* application and termination of the *dBASE Plus* runtime for Web based applications. This property is relevant only for applications built using the BUILD command's WEB parameter.

themeState

Topic group

Indicates whether themes are in use for the application.

Property of

_app object

Description

_app.themeState() returns one of the following values:

- 0 - themes are not available (old OS, or no manifest file, or themes were disabled at the application initialization in another manner).
- 1 - themes are enabled, but with the old theme (the user has chosen the "Windows Classic" theme).
- 2 - themes are enabled and a newer theme is active.

Note:

- With theme states 0 and 1, the controls have the old look and are backward compatible.
- With state 2, some form components may have new colors or their border may have new shape and position. Therefore, with state 2 some modifications (in dBL code) may be needed.

trackRight

Topic group Related topics

Determines if the user can select a popup menu item with a right mouse click.

Property of

Popup

Description

When the *trackRight* property is set to *true* (the default), users can select popup menu items with either the right mouse button or the left mouse button.

Set the *trackRight* property to *false* if you don't want users to be able to select items from a popup menu with a right mouse click.

uncheckedBitmap

Topic group Related topics

A bitmap to display when a menu item is not *checked*.

Property of

Menu

Description

Use *uncheckedBitmap* to display a bitmap when a menu item's *checked* property is *false*. If no bitmap is specified, nothing is displayed when a menu item is not *checked*.

The *uncheckedBitmap* setting can take one of two forms:

RESOURCE <resource id> <dll name>
specifies a bitmap resource and the DLL file that holds it.

FILENAME <filename>
specifies a bitmap file. See class Image for a list of bitmap formats supported by *dBASE Plus*.

Note

The display area in the menu item is very small (13 pixels square with Small fonts). If the bitmap is too large, the top left corner is displayed. Also, the color of the bitmap when the menu item is highlighted changes, depending on the system menu highlight color. Therefore, you may want to limit yourself to simple monochrome bitmaps.

useUACPaths

Topic group

Indicates whether or not dBASE or a dBASE application should create and maintain private copies of various files and folders for each user according to Window's User Account Control (UAC) rules.

Property of

_app object

Background - What is UAC (User Account Controls)

Starting with Vista (and subsequently Windows 7) Microsoft implemented the *User Account Controls* security component. UAC was implemented to make sure applications adhere to certain conventions to make it easier for Windows to prevent program files from being modified or corrupted - whether by user error or by malicious users or by viruses.

Window's UAC rules are intended to:

- Protect installed program files from being modified or damaged by users or programs that should not have access to them.
- Keep each user's files, configuration settings, etc. separate from other users except where shared files or settings are needed.
- Restrict access to any machine wide settings to the maximum extent possible. By default, only users with Administrator privileges have access to machine wide settings.

Windows Vista and Windows 7 implement these rules by carefully limiting default permissions on folders under the Program Files folder tree, the ProgramData folder tree, the Windows folder tree, and the Users folder tree.

Permissions to registry keys are also carefully limited so that standard users will not be allowed to modify any settings that can affect other users.

In order to follow UAC rules a program must:

- place executable code under the Program Files folder tree and NOT attempt to modify or create any new files under this folder tree while running the program. (Standard users generally have read and execute permissions to files under this folder tree. However, programs may be configured to require administrator privileges which would prevent standard users from running them).

- place shared configuration and data files under the ProgramData folder tree - but NOT attempt to modify or create any new files under this folder tree while running the program. (By default, standard users have readonly access to this folder tree).

- place master copies of files needed by each user under the ProgramData folder tree (to be copied to each user's private folder tree).

- setup a private folder tree under the Users folder tree for each user when a user first runs the program so that each user can modify their private files however they wish without interferring with other users.

What does this mean for dBASE Plus

the _app.useUACPaths property gives dBASE Plus the ability to adhere to the Vista and Windows 7 conventions with regard to:

- The location where dBASE Plus creates and stores its .ini files (containing user settings)
- The location where Source and BDE Aliases are pointing
- The location where dBASE Plus creates and stores samples files.
- Where dBASE instructs the BDE to create its temporary files and the default path for the default session at dBASE startup.

During startup, dBASE and dBASE applications will determine the setting for useUACPaths (as described below) and then check useUACpaths to determine the location for various files and folders mentioned above.

When useUACPaths = False, dBASE will NOT use UAC folders and all locations for .ini files, conversion utilities, include files and various sample files are set to subdirectories under the root dBASE directory.

When useUACPaths = True, dBASE will set the locations for .ini files, conversion utilities, include files, various sample files, and temporary files to folders within the current user's private folder tree as required by Window's UAC rules. dBASE determines the path for the user's private folder tree by retrieving the following Window's special folder paths: CSIDL_LOCAL_APPDATA and CSIDL_APPDATA.

How is _app.useUACPaths determined.

when opening Plus.exe or an application (which uses Plusrun.exe) the _app.useUACPaths property is determined in the following manner:

During startup of plus.exe:

- Checks if registry key HKEY_LOCAL_MACHINE\SOFTWARE\dBASE\PLUS\series1\useUACPaths exists and is set to "y" or "Y".

If yes,

Sets _app.useUACPaths to TRUE

Otherwise,

Sets _app.useUACPaths to FALSE

- Next, checks if -v switch was passed to plus.exe on the command line which would override default setting above.

If yes,

If -v1 (or -V1) found on command line,

Sets _app.useUACPaths to TRUE

If -v0 (or -V0) found on command line,

Sets _app.useUACPaths to FALSE

In summary for plus.exe:

If a useUACPaths registry key exists for dBASE Plus, it sets the default for _app.useUACPaths. If this key does NOT exist, _app.useUACPaths is defaulted to FALSE.

If a -v switch is passed on the command line, it will override the default set above.

During startup of plusrun.exe

- Check if registry key HKEY_LOCAL_MACHINE\SOFTWARE\dBASE\PLUS\series1\useUACPaths exists and is set to "y" or "Y".

If yes,

Sets _app.useUACPaths to TRUE

Otherwise,

Sets _app.useUACPaths to FALSE

- Next, checks if -v switch was passed to plusrun.exe (or an application .exe) on the command line which would override default setting above.

If yes,

If -v1 (or -V1) found on command line,

Sets _app.useUACPaths to TRUE

If -v0 (or -V0) found on command line,

Sets _app.useUACPaths to FALSE

- If no -v switch was passed via command line plusrun.exe checks the registry for an application specific registry key specifying the setting for useUACPaths:

HKEY_LOCAL_MACHINE\SOFTWARE\dBASE\PLUS\RuntimeApps\<app file name>\useUACPaths

If this key is found, it overrides the default setting via the above dBASE\PLUS\series1 registry key

If its set to y or Y

Sets _app.useUACPaths to TRUE

If its set to n or N

Sets _app.useUACPaths to FALSE

- If no -v switch was passed via the command line AND no RuntimeApps registry key setting was found for the application, then the application .exe is checked to see if has an embedded setting to set _app.useUACPaths to TRUE.

If an embedded UAC flag is found

Sets _app.useUACPaths to TRUE

In summary for an application .exe:

If a useUACPaths registry key exists for dBASE Plus, its used to set the default for app.useUACPaths. If this key does NOT exist, _app.useUACPaths is defaulted to FALSE.

If an application .exe is built with an embedded UAC setting, the embedded setting will override the global default set for dBASE Plus above.

If an application specific registry key exists, its setting will override the dBASE Plus registry key setting and the embedded UAC setting in the application .exe (if one exists).

If a -v switch is passed on the command line, it will override all of the above settings.

Folders and files affected by _app.useUACPaths

During startup, dBASE PLUS or dBASE PLUS Runtime checks _app.useUACPaths for the following items:

1. **INI file:** Determines default location for plus.ini or an application .ini file

When useUACPaths is true the dBASE Plus plus.ini or an application .ini file will load from a user's local folder tree rather than from the same folder as plus.exe or an application .exe was launched.

First Plus.exe or plusrun.exe will ensure that the following folders exist and create them if they do not yet exist:

For PLUS.EXE:

<CSIDL_LOCAL_APPDATA>\<dBASE Plus subpath>

<CSIDL_APPDATA>\<dBASE Plus subpath>

For an application .exe:

<CSIDL_LOCAL_APPDATA>\<application launch subpath>

<CSIDL_APPDATA>\<application launch subpath>

The ***<... subpath>*** For Both dBASE PLUS IDE and a dBASE Plus Application are evaluated as follows...

- If the launch path contains either "\Program Files" or "\Program Files (x86)", the launch subpath is set to the portion of the launch path remaining, once the portion containing the "\Program Files" or "\Program Files (x86)" folder is removed.

For Example:

If dBASE plus is installed in folder:

C:\Program Files\dBASE\Plus

the <dBASE Plus subpath> will be set to:

\dBASE\Plus

If an application .exe is installed in folder:

C:\Program Files\YourCompany\YourApp

the <application launch subpath> will be set to:

\YourCompany\YourApp

- If the launch path does NOT contain "\Program Files" or "Program Files (x86)", then the <.... subpath> is set to the path remaining after removing the drive letter and colon or removing the top level UNC path from the launch path.

For Example:

If dBASE Plus is launched from:
C:\dBASE\PLUS
The <application launch subpath> is set to:
\dBASE\PLUS
An application built by dBASE Plus is launched from:
\\YourUNCPath\YourCompany\YourApp
The <application launch subpath> is set to:
\YourCompany\YourApp

Next plus.exe or plusrun.exe will ensure an .ini file is loaded as follows:
Check for ini file in a path passed via -C command line switch
If found, load .ini from this path
If not found,
Check for .ini in <CSIDL_LOCAL_APPDATA>\<.... subpath>[\Bin](for dBASE Plus.ini)
If not found,
Check for .ini in <CSIDL_COMMON_APPDATA>\<.... subpath>[\Bin](for dBASE Plus.ini)
if found, copy to <CSIDL_LOCAL_APPDATA>\<.... subpath>[\Bin](for dBASE Plus.ini)
if not found,
Check for .ini in Exe launch path
If found, copy to <CSIDL_LOCAL_APPDATA>\<.... subpath>[\Bin](for dBASE Plus.ini)
If not found,
Create new plus.ini file in:
<CSIDL_LOCAL_APPDATA>\<.... subpath>[\Bin](for dBASE Plus.ini)
The open .ini is now located in <CSIDL_LOCAL_APPDATA>\<.... subpath>[\Bin](for dBASE Plus.ini)
(NOTE: if useUACPaths is false the ini file is saved in the same location as the Plus.exe or application's exe)

2. **Startup and User files**: on Startup dBASE determines whether or not this is the first time a user has launched dBASE Plus. If so a new utility (:dBStartup:InitNewUser.pro) is called to setup the new user.
   In dBASE Plus, InitNewUser.pro will create private copies of the various converter and sample folders as follows:

Converters
dBLClasses
Include
Media
Samples
Web

In a dBASE application, a custom InitNewUser.pro can be used to perform custom setup tasks for a new user.

(NOTE: if useUACPaths is false this entire step is skipped since no user files need to be created and these folders were installed under the root dBASE folder such as C:\Program Files\dBASE\PLUS)

3. **Source Aliases:** dBASE Sets Source Alias paths to match the private locations for the following Source Aliases:

DOS5Conv
Examples
FormControls
Forms

Images
Movies
Nonvisual
Output
ReportControls
Reports
Samples
Toolbars
Webwizards

(NOTE: if useUACPaths is false the source aliases are set to the non-UAC location for these files under the root dBASE folder such as C:\Program Files\dBASE\PLUS)

4. **BDE Aliases:** Determines the BDE paths matching the private locations for the following aliases and creates User Aliases in the user's Plus.ini file pointing to the user's sample folders:

dBASESamples
dBASEContax
dBASESignup
dBASETemp

See the INI topic for more information on User BDE Aliases.

(NOTE: if useUACPaths is false permanent BDE Aliases are created in the IDAPI.CFG file and set to the non-UAC location for these files under the root dBASE folder. e.x.: C:\Program Files\dBASE\PLUS\Samples)

5. **Project Explorer:** Used by Project Explorer to determine the location of:

projexp.ini
Project Explorer temporary files

(NOTE: if useUACPaths is false the projexp.ini file can be found under in the Project Explorer root folder such as C:\Program Files\dBASE\PLUS\BIN\dBLCore\ProjExp)

WAIT

Topic group Related topics

Pauses the current program, displays a message in the results pane of the Command window, and resumes execution when any key is pressed. The keystroke may be stored in a variable.

Syntax

WAIT [<prompt expC>] [TO <memvar>]

<prompt expC>

A character expression that prompts the user for input. If you don't specify <prompt expC>, *dBASE Plus* displays "Press any key to continue..." when you issue WAIT.

TO <memvar>

Assigns a single character to the memory variable you specify for <memvar> as a character-type variable. If <memvar> doesn't exist, *dBASE Plus* creates it. If <memvar> does exist, WAIT overwrites it.

Description

Use WAIT to halt program execution temporarily. Pressing any key exits WAIT and resumes program execution. WAIT is usually used during application development to display information and create simple breakpoints. It is usually not used in deployed applications.

In simple test applications, you can also WAIT to get a single key or character. If the user presses Enter without typing any characters, WAIT assigns an empty string ("") to <memvar>.

Note

If SET ESCAPE is ON, pressing Esc at the WAIT prompt causes *dBASE Plus* to interrupt program execution. If SET ESCAPE is OFF, pressing Esc in response to WAIT causes program execution to resume the same as any other key.

web

Topic group Related topics

Indicates whether the application .exe was built using the WEB parameter

Property of
_app object

Description

The web property provides a means to determine whether the WEB parameter was used when an application was built. Compiling an application by including the BUILD command's WEB parameter allows it to load faster and use fewer resources than a non-WEB application. Additionally, when a web application .exe is run directly, rather than as a parameter to PLUSrun.exe, using the WEB parameter allows it to detect when it's been prematurely terminated by a Web server (as happens when an application takes too long to respond). If a premature termination occurs, PLUSrun.exe also terminates to prevent it from becoming stranded in memory.

A timeout value, the number of milliseconds it takes to remove an orphaned PLUSrun.exe from memory, can be set through the _app objects's *terminateTimerInterval* property.

windowMenu

Topic group Related topics

Specifies a menu object that displays a list of all open MDI windows.

Property of

MenuBar

Description

The *windowMenu* property contains a reference to a menu object that has a MenuBar as its parent. When users open this menu object, *dBASE Plus* displays a pulldown list of all open MDI windows.

The *windowMenu* property automatically places a separator line on the pulldown list between any menu prompts and the list of open windows. The currently active window shows a check next to the window name.

If you use the Menu Designer to create a MenuBar, the *windowMenu* propety is automatically set to an item named Window on the menubar:

```
this.windowMenu = this.Window
```

Core Language

Core language overview

Topic group

This section of the Help file describes the core features of the dBL programming language, primarily:

- Structural elements
- Function linking/loading
- Program flow
- Variable scoping
- Global properties and methods

Basic understanding of programming concepts such as loops and variables is assumed.

class Designer

Topic group Related topics

An object that provides access to the Inspector, Source Editor and streaming engine.

Syntax

[<oRef> =] new Designer([<object>] [,<filename expC>])

<oRef>

A variable or property in which to store a reference to the newly created Designer object.

<object>

The object currently being designed

<filename expC>

The name of the file to which the designed object will be saved.

Properties

The following tables list the properties, events, and methods of interest in the Designer class.

Property	Default	Description
baseClassName	DESIGNER	Identifies the object as an instance of the Designer class
className	(DESIGNER)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
custom	false	Whether the designed object will have a custom keyword
filename	Empty string	The name of the file to which the object's class definition is saved. This should be set before the SAVE method is called
inspector	false	Whether the Designer's inspector is displayed
object	Null	The object currently being designed
selection	Null	The currently selected object displayed in the inspector
sourceChanged	false	Whether a change has been made to the object class by the source editor

unsaved	false	Whether changes have been saved

Event	Parameters	Description
beforeStream		Just before editor calls the designer object to stream code into the editor.
onMemberChange	<expC>	After a change has been made to a member- property, event or method-of the currently selected object in the Inspector. The parameter, <expC>, is the name of the property, event or method.
onNotify	<source name expC>, <filename expC>	When notification is received from another object. Currently, this event fires when the Table Designer, or SQL Designer closes, and the first parameter is, "TABLE_DESIGNER_CLOSE", or "SQL_DESIGNER_CLOSE". The second parameter is the *filename* that was being designed.
onSelChange		After a different object has been selected in the inspector and the selection property modified.

Method	Parameters	Description
editor( )		Opens a source editor to display the current object.
isInherited( )	<oRef1>, <oRef2>	Determines if an object, <oRef2>, in the designer <oRef1> is inherited from a superclass. By doing so, the *isInherited*( ) method can be used to programatically enforce rules of inheritance.
loadObjectFromFile( )	<filename expC>	Loads the object property of an existing file. Resets the *filename* property to <filename expC>.
reloadFromEditor( )		Reloads the object from the current editor contents. Resets the *sourceChanged* property.
save( )		Saves the current object to *filename.*
update( )		Causes the source editor to reflect changes made to an object or any of it's components.

Description

Use Designer objects to gain access to the Inspector, Source editor or streaming engine during RunMode. While the Designer's parameters, OBJECT and FILENAME, are listed as optional, they must be used in certain situations.

When modifying a custom class, the filename parameter must be used to specify the file from which the object was loaded. The filename parameter is not necessary when a new class is being derived from the custom class.

When creating a new custom class from a base class, the filename parameter is optional. However, if no parameters are specified, the Designer must subsequently be intialized using it's properties and/or methods.

When designing a new class, the OBJECT and FILENAME parameters must be set.

When modifying an existing class, the *loadObjectFromFile* ( ) method must be called.

class Exception

Topic group Related topics Example

An object that describes an exception condition.

Syntax

[<oRef> =] new Exception( )

<oRef>

A variable or property in which to store a reference to the newly created Exception object.

Properties

The following table lists the properties of the Exception class. (No events or methods are associated with this class.)

Property	Default	Description
baseClassName	EXCEPTION	Identifies the object as an instance of the Exception class
className	(EXCEPTION)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
code	0	A numeric code to identify the type of exception
filename		The name of the file in which a system-generated exception occurs
lineNo	0	The line number in the file in which a system-generated exception occurs
message		Text to describe the exception

Description

An Exception object is automatically generated by *dBASE Plus* whenever an error occurs. The object's properties contain information about the error.

You can also create an Exception object manually, which you can fill with information and THROW to manage execution or to jump out of deeply nested statements.

You may subclass the Exception class to create your own custom exception objects. A TRY block may be followed by multiple CATCH blocks, each one looking for a different exception class.

class Object

Topic group Related topics Example

An empty object.

Syntax

[<oRef> =] new Object( )

<oRef>

A variable or property in which to store a reference to the newly created object.

Properties

An object of the Object class has no initial properties, events, or methods.

Description

Use the Object class to create your own simple objects. Once the new object is created, you may add properties and methods through assignment. You cannot add events.

This technique of adding properties and methods on-the-fly is known as dynamic subclassing. In *dBASE Plus*, dynamic subclassing supplements formal subclassing, which is achieved through CLASS definitions.

The Object class is the only class in *dBASE Plus* that does not have the read-only *baseClassName* or *className* properties.

ARGCOUNT( )

Topic group Related topics Example

Returns the number of parameters passed to a routine.

Syntax

ARGCOUNT( )

Description

Use ARGCOUNT( ) to determine how many parameters, or arguments, have been passed to a routine. You may alter the behavior of the routine based on the number of parameters. If there are fewer parameters than expected, you may provide default values.

ARGCOUNT( ) returns 0 if no parameters are passed.

The function PCOUNT( ) is identical to ARGCOUNT( ). Neither function recognizes parameters passed to codeblocks. If called within a codeblock, the function will return the parameter information for the currently executing FUNCTION or PROCEDURE.

ARGVECTOR( )

Topic group Related topics Example

Returns the specified parameter passed to a routine.

Syntax

ARGVECTOR(<parameter expN>)

<parameter expN>

The number of the parameter to return. 1 returns the first parameter, 2 returns the second parameter, etc.

Description

Use ARGVECTOR( ) to get a copy of the value of a parameter passed to a routine. Because it is a copy, there is no danger of modifying the parameter, even if it was a variable that was passed by reference. For more information on parameter passing, see PARAMETERS.

ARGVECTOR( ) can be used in a routine that receives a variable number of parameters, where declaring the parameters would be difficult. ARGVECTOR( ) cannot be used within a codeblock.

baseClassName

Topic group Related topics

Identifies to which class an object belongs.

Property of

All classes except Object.

Description

The *baseClassName* property identifies the class constructor that originally created the object. Although you may dynamically subclass the object by adding new properties, the *baseClassName* property does not change.

The *baseClassName* property is read-only.

CASE

Topic group

Designates a block of code in a DO CASE block.

Description

See DO CASE for details.

CATCH

Topic group

Designates a block of code to execute if an exception occurs inside a TRY block.

Description

See TRY...ENDTRY for details.

CLASS

Topic group

A class declaration including constructor code, which typically creates member properties, and class methods.

Syntax

```
CLASS <class name>[(<parameters>)]
 [OF <superclass name>[(<parameters>)]
  [CUSTOM]
  [FROM <filename expC>] ]

 [PROTECT <propertyList>]
 [<constructor code>]
 [<methods>]
ENDCLASS
```

<class name>

The name of the class. Although dBASE Plus imposes no limit to the length of class names, it recognizes only the first 32 characters.

OF <superclass name>

Indicates that the class is a derived class that inherits the properties defined in the superclass. The superclass constructor is called before the <constructor code> in the current CLASS is called, which means that any properties created in the superclass are inherited by the class.

<parameters>

Optional parameters to pass to the class, and through to the superclass.

CUSTOM

Identifies the class as a custom component class, so that its predefined properties are not streamed out by the visual design tools.

FROM <filename>

<filename> specifies the file containing the definition code for the <superclass>, if the <superclass> is not defined in the same file as the class.

PROTECT <propertyList>

<propertyList> is a list of properties and/or methods of the class which are to be accessible only by other members of the class, and by classes derived from the class.

<constructor code>

The code that is called when a new instance of the class is created with the NEW operator or a DEFINE statement. The constructor consists of all the code at the top of the class declaration up to the first method.

<methods>

Any number of functions designed for the class.

ENDCLASS

A required keyword that marks the end of the CLASS structure.

Description

Use CLASS to create a new class.

A class is a specification, or template, for a type of object. *dBASE Plus* provides many stock classes, such as Form and Query; for example, when you create a form, you are creating a new Form object that has the standard properties and methods from the Form class. However, when you declare a class with CLASS, you specify the properties and methods of objects derived from the new class.

A CLASS declaration formalizes the creation of an object and its methods. Although you can always add properties to an object and assign methods dynamically, a CLASS simplifies the task and allows you to build a clear class hierarchy.

Another benefit is polymorphism. Every FUNCTION (or PROCEDURE) defined in the CLASS becomes a method of the class. An object of that class automatically has a property with the same name as each FUNCTION that contains a reference to that FUNCTION. Because a method is part of the CLASS, different functions may use the same name as long as they are methods of different classes. For example, you can have multiple copy( ) functions in different classes, with each one applying to objects of that class. Without classes, you would have to name the functions differently even if they performed the same task conceptually.

Before the first statement in the constructor is executed, if the CLASS extends another class, the constructor for that superclass has already been executed, so the object contains all the superclass properties. Any properties that refer to methods, as described in the previous paragraph, are assigned. This means that if the CLASS contains a method with the same name as a method in a superclass, the method in the CLASS overrides the method in the superclass. The CLASS constructor, if any, then executes.

In the constructor, the variable *this* refers to the object being created. Typically, the constructor creates properties by assigning them to *this* with dot notation. However, the constructor may contain any code at all, except another CLASS—you can't nest classes—or a FUNCTION, since that FUNCTION would become a method of the class and indicate the end of the constructor.

Properties and methods can be protected to prevent the user of the class from reading or changing the protected property values, or calling the protected methods from outside of the class.

className

Topic group Related topics

Identifies an object as an instance of a custom class. When no custom class exists, the *className* property defaults to the *baseClassName*.

Property of

All classes except Object.

Description

The *className* property identifies a custom object derived from a standard *dBASE Plus* class. The *className* property is read-only.

CLEAR MEMORY

Topic group Related topics

Clears all user-defined memory variables.

Syntax

CLEAR MEMORY

Description

Use CLEAR MEMORY to release all memory variables (except system memory variables), including those declared PUBLIC and STATIC and those initialized in higher-level routines. CLEAR MEMORY has no effect on system memory variables.

Note

CLEAR MEMORY does not explicitly release objects. However, if the only reference to an object is in a memory variable, releasing the variable with CLEAR MEMORY will in turn release the object.

Issuing RELEASE ALL in the Command window has the same effect as CLEAR MEMORY. However, issuing RELEASE ALL in a program clears only memory variables created at the same program level as the RELEASE ALL statement, and has no effect on higher-level, public, or static variables. CLEAR MEMORY, whether issued in a program or in the Command window, always has the same effect, releasing all variables.

To clear only selected memory variables, use RELEASE.

CLEAR PROGRAM

Topic group Related topics

Clears from memory all program files that aren't currently executing and aren't currently open with SET PROCEDURE or SET LIBRARY.

Syntax

CLEAR PROGRAM

Description

Program files are loaded into memory when they are executed with DO, and when they are loaded as library or procedure files with SET LIBRARY and SET PROCEDURE. When *dBASE Plus* is done with the program—the execution is complete, or the file is unloaded—the program file is not automatically cleared from memory. This allows these files to be quickly reloaded without having to reread them from disk. *dBASE Plus'* internal dynamic memory management will clear these files if it needs more memory; for example, when you create a very large array.

You may use CLEAR PROGRAM to force the clearing of all inactive program (object code) files from memory. The command doesn't clear files that are currently executing or files that are currently open with SET PROCEDURE or SET LIBRARY. However, if you close a file (for example, with CLOSE PROCEDURE), a subsequent CLEAR PROGRAM clears the closed file from memory.

CLEAR PROGRAM is rarely used in a deployed application. Because of the event-driven nature of *dBASE Plus*, program files must remain open to handle events; these files are not affected by CLEAR PROGRAM anyway. Also, the amount of memory used by dormant program files is small compared to the total amount of memory available. You are more likely to use CLEAR PROGRAM during development, for example to ensure that you are running the latest version of a program file, and not one that is stuck in memory.

CLOSE PROCEDURE

Topic group Related topics

Closes one or more procedure files, preventing further access and execution of its functions, classes, and methods.

Syntax

CLOSE PROCEDURE [<filename list>] | [PERSISTENT]

<filename list>

A list of procedure files you want to close, separated by commas. If you specify a file without including its extension, *dBASE Plus* assumes PRG. If you omit <filename list>, all procedure files not tagged PERSISTENT are closed, regardless of their load count.

PERSISTENT

When <filename list> is omitted, CLOSE PROCEDURE PERSISTENT will close all files, including those tagged PERSISTENT. Without the PERSISTENT designation, these files would not be affected.

Description

CLOSE PROCEDURE reduces the load count of each specified program file by one. If that reduces its load count to zero, then that program file is closed, and its memory is marked as available for reuse.

When you specify more than one file in <filename list>, they are processed in reverse order, from right to left. If a specified file is not open as a procedure file, an error occurs, and no more files in the list are processed.

Closing a program file does not automatically remove the file from memory. If a request is made to open that program file, and the file is still in memory and its source code has not been updated, it will be reopened without having to reread the file from disk. Use CLEAR MEMORY to release a closed program file from memory.

In a deployed application, it is not unusual to open program files as procedure files and never close them. Because of the event-driven nature of *dBASE Plus*, program files must remain open to respond to events. The memory used by a procedure file is small in comparison to the amount of system memory.

See SET PROCEDURE for a description of the reference count system used to manage procedure files. You may issue SET PROCEDURE TO or CLOSE PROCEDURE with no

<filename list> to close all open procedure files, not tagged PERSISTENT, regardless of their load count.

DEFINE

Topic group

Creates an object from a class.

Syntax

DEFINE <class name> <object name>
[OF <container object>]
[FROM <row, col> TO <row, col> > | <AT <row, col>]
[PROPERTY <stock property list>]
[CUSTOM <custom property list>]

<class name>

The class of the object to create.

<object name>

The identifier for the object you create. <object name> will become an object reference variable, or a named property of the container if a <containter object> is specified.

OF <container object>

Identifies the object that contains the object you define.

FROM <row>, <col> TO <row>, <col> | AT <row>, <col>

Specifies the initial location and size of the object within its container. FROM and TO specify the upper left and lower right coordinates of the object, respectively. AT specifies the position of the upper left corner.

PROPERTY <stock property list>

Specifies values you assign to the built-in properties of the object.

CUSTOM <custom property list>

Specifies new properties you create for the object and the values you assign to them.

Description

Use DEFINE to create an object in memory. DEFINE provides an alternate, shorthand syntax for creating objects that directly maps to using the NEW operator. The equivalence depends on whether the object created with DEFINE is created inside a container object. With no container,

```
define <class name> <object name>
```

is equivalent to:

```
<object name> = new <class name>( )
```

With a container,

```
define <class name> <object name> of <container object>
```

is equivalent to:

```
new <class name>( <container object>, "<object name>" )
```

where <object name> becomes an all-uppercase string containing the specified name. These two parameters, the container object reference and the object name, are the two properties expected by the class constructors for all stock control classes such as PushButton and Entryfield. For example, these two sets of statements are functionally identical (and you can use the first statement in one set with the second statement of the other set):

```
define Form myForm
define PushButton cancelButton of myForm
myForm = new Form( )
new PushButton( myForm, "CANCELBUTTON" )
```

The FROM or AT clause of the DEFINE command provide a way to specify the *top* and *left* properties of an object, and the TO coordinates are used to calculate the object's *height* and *width*.

The PROPERTY clause allows assignment to existing properties only. Attempting to assign a value to a non-existent property generates an error at runtime. This will catch spelling errors in property names, when you want to assign to an existing property; it prevents the creation of a new property with the misspelled name. Using the assignment-only (:=) operator has the same effect when assigning directly to a property in a separate assignment statement. In contrast, the CUSTOM clause will create the named property if it doesn't already exist.

While the DEFINE syntax offers some amenities, it is not as flexible as using the NEW operator and a WITH block. In particular, with DEFINE you cannot pass any parameters to the class constructor other than the two properties used for control containership, and you cannot assign values to the elements of properties that are arrays.

DO

Topic group Related topics

Runs a program or function.

Syntax

DO <filename> | ? | <filename skeleton> |
 <function name>
[WITH <parameter list>]

<filename> | ? | <filename skeleton>

The program file to execute. The ? and <filename skeleton> options display a dialog box from which you can select a file. If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory, then in the search path in search order. See "Search path and order" later in this section for more information.

If you specify a file without including its extension, *dBASE Plus* assumes a .PRO extension (a compiled object file). If *dBASE Plus* can't find a .PRO file, it looks for a .PRG file (a source file), which, if found, it compiles. By default, *dBASE Plus* creates the .PRO in the same directory as the .PRG, which might not be the current directory.

<function name>

The function name in an open program file to execute. The function must be in the program file containing the DO command that calls it, or in a separate open file on the search path. The search path is described later in this section.

WITH <parameter list>

Specifies memory variable values, field values, or any other valid expressions to pass as parameters to the program or function. See the description of PARAMETERS for information on parameter passing.

Description

Use DO to run program files from the Command window or to run other programs from a program. If you enter DO in the Command window, control returns to the Command window

when the program or function ends. If you use DO in a program to execute another program, control returns to the program line following the DO statement when the program ends.

Although you may use DO to execute functions, common style dictates the use of the call operator (the parentheses) when calling functions, and the DO command when running a program file. The DO command supports the use of a file path and extension, and the ? and <filename skeleton> options. The call operator supports calling a function by name only. In the not-recommended situation where you have a program file that has the same name as a function loaded into memory, the DO command will execute the program file, and the call operator will executed the loaded function. Other than these differences, the two calling methods behave the same, and follow the same search rules described later in this section.

You may nest routines; that is, one routine may call another routine, which may call another routine, and so on. This series of routines, in the order in which they are called, is referred to as the call chain.

When *dBASE Plus* executes or loads a program file, it will automatically compile the program file into object code when either:

There is no object code file, or

SET DEVELOPMENT is ON, and program file is newer than the object code file (the source code file's last update date and time is later than the object code file's)

When *dBASE Plus* encounters a function call in a program file, it looks in that file for a FUNCTION or PROCEDURE of the specified name. If the current program file contains a FUNCTION and a PROCEDURE with the same name, *dBASE Plus* executes the first one declared. If *dBASE Plus* doesn't find a FUNCTION or PROCEDURE definition of the specified name in the same program file, it looks for a program file, FUNCTION, or PROCEDURE of the specified name on the search path in search order.

Search path and order

If the name you specify with DO doesn't include a path or a file-name extension, it can be a file, FUNCTION, or PROCEDURE name. To resolve the ambiguity, *dBASE Plus* searches for the name in specific places (the search path) in a specific order (the search order) and runs the first program or function of the specified name that it finds. The search path and order *dBASE Plus* uses is as follows:

1. The executing program's object file (.PRO)
    - Other open object files (.PRO) in the call chain, in most recently opened order
    - The file specified by SYSPROC = <filename> in PLUS.ini
    - Any files opened with SET PROCEDURE, SET PROCEDURE...ADDITIVE, or SET LIBRARY statements, in the order in which they were opened
    - The object file (.PRO) with the specified name in the search path
    - The program file (.PRG) with the specified name in the search path, which*dBASE Plus* automatically compiles

The search path is controlled with the SET PATH command. It is not used when you are running a compiled EXE (a deployed application)—all program files must be linked into the executable. All path information is lost during linking and ignored during execution, which means that you cannot have more than one file with the same name, even if they originally came from different directories.

Because program files must be compiled into object code to be linked into a compiled EXE, the last search step, #6, does not apply when running a compiled EXE.

DO CASE

Topic group Related topics Example

Conditionally processes statements by evaluating one or more conditions and executing the statements following the first condition that evaluates to true.

Syntax

```
DO CASE
CASE <condition expL 1>
  <statements>
[CASE <condition expL 2>
  <statements>...]
[OTHERWISE
  <statements>]
ENDCASE
```

CASE <condition expL>

If the condition is true, executes the set of commands between CASE and the next CASE, OTHERWISE, or ENDCASE command, and then transfers control to the line following ENDCASE. If the condition is false, control transfers to the next CASE, OTHERWISE, or ENDCASE command.

<statements>

Zero or more statements to execute if the preceding CASE statement evaluates to *true*.

OTHERWISE

Executes a set of statements if all the CASE statements evaluate to *false*.

ENDCASE

A required keyword that marks the end of the DO CASE structure.

Description

DO CASE is similar to IF...ELSE...ENDIF. As with IF conditions, *dBASE Plus* evaluates DO CASE conditions in the order they're listed in the structure. DO CASE acts on the first true condition in the structure, even if several apply. In situations where you want only the first true instance to be processed, use DO CASE instead of a series of IF commands.

Also, use DO CASE when you want to program a number of exceptions to a condition. The CASE <condition> statements can represent the exceptions, and the OTHERWISE statement the remaining situation.

Starting with the first CASE condition, *dBASE Plus* does the following.

- Evaluates each CASE condition until it encounters one that's *true*
- Executes the statements between the first true CASE statement and the next CASE, OTHERWISE, or ENDCASE (if any)
- Exits the DO CASE structure without evaluating subsequent CASE conditions
- Moves program control to the first line after the ENDCASE command

If none of the conditions are true, *dBASE Plus* executes the statements under OTHERWISE if it's included. If no OTHERWISE statement exists, *dBASE Plus* exits the structure without executing any statements and transfers program control to the first line after the ENDCASE command.

DO CASE is functionally identical to an IF...ELSEIF...ENDIF structure. Both specify a series of conditions and an optional fallback (OTHERWISE and ELSE) if none of the conditions are *true*. Common style dictates the use of DO CASE when the conditions are dependent on the same variable, for example what key was pressed, while IF...ELSEIF...ENDIF is used when the conditions are not directly related. In addition, DO CASE usually involves more indenting of code.

DO WHILE

Topic group Related topics Example

Executes the statements between DO WHILE and ENDDO while a specified condition is *true*.

Syntax

```
DO WHILE <condition expL>
 [ <statements> ]
ENDDO
```

<condition expL>

A logical expression that is evaluated before each iteration of the loop to determine whether the iteration should occur. If it evaluates to *true*, the statements are executed. Once it evaluates to *false*, the loop is terminated and execution continues with the statement following the ENDDO.

<statements>

Zero or more statements executed in each iteration of the loop.

ENDDO

A required keyword that marks the end of the DO WHILE loop.

Description

Use a DO WHILE loop to repeat a statement or block of statements while a condition is *true*. If the condition is initially *false*, the statements are never executed.

You may also exit the loop with EXIT, or restart the loop with LOOP.

DO...UNTIL

Topic group Related topics Example

Executes the statements between DO and UNTIL at least once while a specified condition is false.

Syntax

```
DO
 [ <statements> ]
UNTIL <condition expL>
```

<statements>

Zero or more statements executed in each iteration of the loop.

UNTIL<condition expL>

The statement that marks the end of the DO...UNTIL loop. <condition expL> is a logical expression that is evaluated after each iteration of the loop to determine whether the iteration should occur again. If it evaluates to *false*, the statements are executed. Once it evaluates to *true*, the loop is terminated and execution continues with the statement following the UNTIL.

Description

Use a DO...UNTIL loop to repeat a block of statements until a condition is *true* (in other words, while the condition is *false*). Because the condition is evaluated at the end of the loop, a DO...UNTIL loop always executes at least once, even when the condition is initially *true*.

You may also exit the loop with EXIT, or restart the loop with LOOP.

DO...UNTIL is rarely used. In most condition-based loops, you don't want to execute the loop at all if the condition is initially invalid. DO WHILE loops are much more common, because they check the condition before they begin.

In a DO WHILE loop, the condition fails—that is, the loop should not be executed—when it evaluates to *false*; in a DO...UNTIL loop, the condition fails when it evaluates to *true*. This is simply the result of the wording of the looping commands. You can easily reverse any logical condition by using the logical NOT operator or the opposite comparison operator (for example, less than instead of greater than or equal, or not equal instead of equal).

ELSE

Topic group

Designates an alternate statement to execute if the condition in an IF statement is *false*.

Description

See IF for details.

ELSEIF

Topic group

Designates an alternate condition to test if the condition in an IF statement is *false*.

Description

See IF for details.

EMPTY( )

Topic group Related topics

Returns *true* if a specified expression is empty.

Syntax

EMPTY(<exp>)

<exp>

An expression of any type.

Description

Use EMPTY( ) to determine if an expression is empty. The definition of empty depends on the type of the expression:

Expression type	Empty if value is
Numeric	0 (zero)
String	empty string ("") or a string of just spaces (" ")
Date	blank date ({ / / })
Logical	*false*
Null	*null* is always considered empty
Object reference	Reference points to object that has been released

Note that event properties that have not been assigned handlers have a value of *null*, and are therefore considered empty. In contrast, an object reference pointing to an object that has been released is not *null*; you must use EMPTY( ).

EMPTY( ) is similar to ISBLANK( ). However, ISBLANK( ) is intended to test field values; it differentiates between zero and blank values in numeric fields, while EMPTY( ) does not. EMPTY( ) understands null values and object references, while ISBLANK( ) does not. For more information, see ISBLANK( ).

ENUMERATE( )

Topic group Related topics Example

Returns a listing of the member names of an object.

Syntax

Enumerate(<oRef>)

<oRef>)

Object reference to any valid object.

Description

Use ENUMERATE( ) to retrieve a listing of the member names of an object with each member name identified as a property, event, or method of the specified object.

ENUMERATE( ) returns an AssocArray object. Each index into the AssocArray is a member name for the enumerated object. The value of the index is filled with one of the following values:

Value	Description
P	The type of member is a property.
E	The type of member is an event.
M	The type of member is a method.

EXIT

Topic group Related topics Example

Immediately terminates the current loop. Execution continues with the statement after the loop.

Syntax

EXIT

Description

Normally, all of the statements in the loop are executed in each iteration of the loop; in other words, the loop always exits after the last statement in the loop. Use EXIT to exit a loop from the middle of a loop, due to some extra or abnormal condition.

In most cases, you don't have to resort to using EXIT; you can code the condition that controls the loop to handle the extra condition. The condition is tested between loop iterations, after the last statement, but that usually means that there are some statements that should not be executed because of this condition. Those statements would have to be conditionalized out with an IF statement. Therefore, often it's simpler to EXIT out of a loop immediately once the condition occurs.

filename

Topic group Related topics

The name of a file containing an existing class definition, or the name of a file to which a newly created class definition will be saved.

Property of

Designer

Description

To design a new custom class, or modify a stock class, set the *filename* property to the name of the file under which the class definition will be saved. While designating a filename, when initially creating a custom class, is not required, a filename must be assigned before the class definition can be saved. Calling the save( ) method, without first setting the *filename* property, will open a Save As dialog.

When modifying an existing custom class, the *filename* property will be set by the *loadObjectFromFile*( ) method.

FINALLY

Topic group

Designates a block of code that always executes after a TRY block, even if an exception occurs.

Description

See TRY...ENDTRY for details.

FINDINSTANCE( )

Topic group Related topics Example

Returns an object of the specified class from the object heap.

Syntax

FINDINSTANCE(<classname expC> [, <previous oRef>])

<classname expC>

The name of the class you want to find an instance of. <classname expC> is not case-sensitive.

<previous oRef>

When omitted, FINDINSTANCE( ) returns the first instance of the specified class. Otherwise, it returns the instance following <previous oRef> in the object heap.

Description

Use FINDINSTANCE( ) to find any instance of a particular class, or to find all instances of a class in the object heap.

Objects are stored in the object heap in no predefined order. Creating a new instance of a class or destroying an instance may reorder all other instances of that class. A newly created object is not necessarily last in the heap.

Sometimes you will want to make sure there is only one instance of a class, and reuse that instance; a particular toolbar is the prime example. To see if there is an instance of that class, call FINDINSTANCE( ) with the class name only. If the return value is *null*, there is no instance of that class in memory.

Other times, you may want to iterate through all instances of a class to perform an action. For example, you may want to close all data entry forms, which are all instances of the same class. Call FINDINSTANCE( ) with the class name only to find the first instance of the class. Then call FINDINSTANCE( ) in a loop with the class name and the object reference to get the next instance in the object heap. When FINDINSTANCE( ) returns *null*, there are no more instances.

FOR...ENDFOR

Topic group Related topics Example

Executes the statements between FOR and ENDFOR the number of times indicated by the FOR statement.

Syntax

```
FOR <memvar> = <start expN> TO <end expN> [STEP <step expN>]
 [ <statements> ]
ENDFOR | NEXT
```

<memvar>

The loop counter, a memory variable that's incremented or decremented and then tested each time through the loop.

<start expN>

The initial value of <memvar>.

<end expN>

The final allowed value of <memvar>.

STEP <step expN>

Defines a step size (<step expN>) by which *dBASE Plus* increments or decrements <memvar> each time the loop executes. The default step size is 1.

When <step expN> is positive, *dBASE Plus* increments <memvar> until it is greater than <end expN>. When <step expN> is negative, *dBASE Plus* decrements <memvar> until it is less than <end expN>.

<statements>

Zero or more statements executed in each iteration of the loop.

ENDFOR | NEXT

A required keyword that marks the end of the FOR loop. You may use either ENDFOR (more dBASE-ish) or NEXT.

Description

Use FOR...ENDFOR to execute a block of statements a specified number of times. When *dBASE Plus* first encounters a FOR loop, it sets <memvar> to <start expN>, and reads the values for <end expN> and <step expN>. (If <end expN> or <step expN> are variables and are changed inside the loop, the loop will not see the change and the original values will still be used to control the loop.)

The loop counter is checked at the beginning of each iteration of the loop, including the first iteration. If <memvar> evaluates to a number greater than <end expN> (or less than <end expN> if <step expN> is negative), *dBASE Plus* exits the FOR loop and executes the line following ENDFOR (or NEXT). Therefore, it's possible that the loop body is not executed at all.

If <memvar> is in the range from <start expN> through <end expN>, the loop body is executed. After executing the statements in the loop, <step expN> is added to <memvar>, and the loop counter is checked again. The process repeats until the loop counter goes out of range.

You may also exit the loop with EXIT, or restart the loop with LOOP.

The <memvar> is usually used inside the loop to refer to numbered items, and continues to exist after the loop is done, just like a normal variable. If you do not want the variable to be the default private scope, you should declare the scope of the variable before the FOR loop.

FUNCTION

Topic group Related topics

Defines a function in a program file including variables to represent parameters passed to the function.

Syntax

```
FUNCTION <function name>[ ( [<parameter list>] ) ]
 [<statements>]
```

<function name>

The name of the function. Although *dBASE Plus* imposes no limit to the length of function names, it recognizes only the first 32 characters.

(<parameter list>)

Variable names to assign to data items (or parameters) passed to the function by the statement that called it. The variables in <parameter list> are local in scope, protecting them from modification in lower-level subroutines. For more information about the local scope, see LOCAL.

<statements>

Any statements that you want the function to execute. You can call functions recursively.

Description

Use functions to create code modules. By putting commonly used code in a function, you can easily call it whenever needed, pass parameters to the function, and optionally return a value. You also create more modular code, which is easier to debug and maintain.

When a FUNCTION is defined inside a CLASS definition, the FUNCTION is considered a method of that CLASS. You cannot nest functions.

The keywords FUNCTION and PROCEDURE are interchangable in *dBASE Plus*.

A single program file can contain a total of 184 functions and methods. Each class also counts as one function (for the class constructor). To access more functions simultaneously, use SET PROCEDURE...ADDITIVE. The maximum size of a function is limited to the maximum size of a program file.

When a function is called via an object, usually as a method or event handler, the variable *this* refers to the object that called the function.

Function naming restrictions

Do not give a function the same name as the file in which it's contained. Statements at the beginning of the file, before any FUNCTION, PROCEDURE, or CLASS statement, are considered to be a function (not counted against the total limit) with the same name as the file. (This function is sometimes referred to as the "main" procedure in the program file.) Multiple functions with the same name do not cause an error, but the first function with that name is the only one that is ever called.

Don't give the function the same name as a built-in dBL function. You cannot call such a function with the DO command, and if you call the function with the call operator (parentheses), *dBASE Plus* always executes its built-in function instead.

Also do not give the function a name that matches a dBL command keyword. For example, you should not name a function OTHER( ) because that matches the beginning of the keyword OTHERWISE. When you call the OTHER( ) function, the compiler will think it's the OTHERWISE keyword and will generate an error, unless you happen to be in a DO CASE block, in which case it will be treated like the OTHERWISE keyword, instead of calling the function.

These function naming restrictions do not apply to methods, because calling a method through the dot or scope resolution operator clearly indicates what is being called. However, you may run into problems calling methods inside a WITH block. See WITH for details.

Making procedures available

You can include a procedure in the program file that uses it, or place it in a separate program file you access with SET PROCEDURE or SET LIBRARY. If you include a procedure in the program file that uses it, you should place it at the end of the file and group it with other procedures.

When you call a procedure, *dBASE Plus* searches for it in the search path in search order. If there is more than one procedure available with the same name, *dBASE Plus* runs the first one it finds. For this reason, avoid using the same name for more than one procedure. See the description of DO for an explanation of the search path and order *dBASE Plus* uses.

IF

Topic group

Conditionally executes statements by evaluating one or more conditions and executing the statements following the first condition that evaluates to *true*.

Syntax

```
IF <condition expL 1>
 [ <statements> ]
[ELSEIF <condition expL 2>
 <statements>
[ELSEIF   <condition expL 3>
 <statements>...]]
[ELSE
 [ <statements> ]]
ENDIF
```

<condition expL>

A logical expression that determines if the set of statements between IF and the next ELSE, ELSEIF, or ENDIF command execute. If the condition is *true*, the statements execute. If the condition is false, control passes to the next ELSE, ELSEIF, or ENDIF.

<statements>

One or more statements that execute depending on the value of <condition expL>.

ELSEIF <condition expL> <statements>

Specifies that when the previous IF or ELSEIF condition is false, control passes to this ELSEIF <condition expL>. As with IF, if the condition is true, only the set of statements between this ELSEIF and the next ELSEIF, ELSE, or ENDIF execute. If the condition is false, control passes to the next ELSEIF, ELSE, or ENDIF.

You can enter this option as either ELSEIF or ELSE IF. The ellipsis (...) in the syntax statement indicates that you can have multiple ELSEIF statements.

ELSE <statements>

Specifies statements to execute if all previous conditions are *false*.

ENDIF

A required keyword that marks the end of the IF structure.

Description

Use IF to evaluate one or more conditions and execute only the set of statements following the first condition that evaluates to *true*. For the first true condition, *dBASE Plus* executes the statements between that program line and the next ELSEIF, ELSE, or ENDIF, then skips everything else in the IF structure and executes the program line following ENDIF. If no condition is true and an associated ELSE command exists, *dBASE Plus* executes the set of statements after ELSE and then executes the program line following ENDIF.

Use IF...ENDIF to test one condition and IF...ELSEIF...ENDIF to test two or more conditions. If you have more than three conditions to test, consider using DO CASE instead of IF. Compare the example in this section with the example for DO CASE.

If you’re evaluating a condition to decide which value you want to assign to a variable or property, you may be able to use the IIF( ) function, which involves less duplication (you don’t have to type the target of the assignment twice).

You can nest IF statements to test multiple conditions; however, the ELSEIF option is an efficient alternative. When you use ELSEIF, you don't need to keep track of which ELSE applies to which IF, nor do you have to put in an ending ENDIF.

You can put many statements for each condition. If the number of statements in a set makes the code hard to read, consider putting them in a function and calling the function from the IF statement instead.

IIF( )

Topic group Related topics

Returns one of two values depending on the result of a specified logical expression.

Syntax

IIF(<expL>, <exp1>, <exp2>)

<expL>

The logical expression to evaluate to determine whether to return <exp1> or <exp2>.

<exp1>

The expression to return if <expL> evaluates to *true*.

<exp2>

The expression to return if <expL> evaluates to *false*. The data type of <exp2> doesn't have to be the same as that of <exp 1>.

Description

IIF( ) stands for "immediate IF" and is a shortcut to the IF...ELSE...ENDIF programming construct. Use IIF( ) as an expression or part of an expression where using IF would be cumbersome or not allowed. In particular, if you're evaluating a condition to decide which value you want to assign to a variable or property, using IIF( ) involves less duplication (you don't have to type the target of the assignment twice).

If <exp1> and <exp2> are *true* and *false*, in either order, using IIF( ) is redundant because <expL> must evaluate to either *true* or *false* anyway.

isInherited( )

Topic group Example

Returns true if the object reference passed in to it refers to an object that is part of a superclass, otherwise, the *isInherited*( ) method returns false.

Syntax

<oRef1>.*isInherited*(<oRef2>)

<oRef1>

An object reference to a designer object

<oRef2>

An object reference to an object contained within the Form, Report, or Datamodule currently loaded into the designer object (oRef1).

Property of

Designer

Description

Use the *isInherited*( ) method to programatically enforce rules of inheritance, such as deleting an inherited Query object from a subclassed dataModule

Take the case of a dataModule (dmd2), subclassed from another dataModule (dmd1), containing Query object 1 and Query object 2, and currently being designed in the Data Module designer.

If Query object 1, currently containted in dmd2, was inherited from its superclass, dmd1, you would not be able to remove it (delete it) from the dataModule dmd2. Should a user attempt such a delete, the *isInherited*( ) method would determine that:

> In the current designer // **Data Module (<oRef1>)**
>
> An object // **Query object 1 (<oRef2>)**
>
> Was inherited from a superClass // **(dmd1)**

The *isInherited*( ) method would return true, and the removal of Query object 1 could be disallowed.

LOCAL

Topic group Related topics

Declares memory variables that are visible only in the routine where they're declared.

Syntax

LOCAL <memvar list>

<memvar list>

The list of memory variables to declare local.

Description

Use LOCAL to declare a list of memory variables available only to the routine in which the command is issued. Local variables differ from those declared PRIVATE in the following ways:

- Private variables are available to lower-level subroutines, while local variables are not. Local variables are accessible only to the routine—the program or function—in which they are declared.
- TYPE( ) does not "see" local variables. If you want to determine the TYPE( ) of a local variable, you must copy it to a private (or public) variable and call TYPE( ) with that variable name in a string.
- You cannot use a local variable for macro substitution with the & operator. Again, you must copy it to a private (or public) variable first.
- LOCAL variables cannot be inspected using the Debugger.

Despite these limitations, local variables are generally preferred over private variables because of their limited visibilty. You cannot accidentally overwrite them in a lower-level routine, which would happen if you forget to hide a public variable; nor can you inadvertently use a variable created in a higher-level routine, thinking that it’s one declared in the current routine, which would happen if you misspell the variable name in the current routine.

Note The special variables *this* and *form* are local.

You must declare a variable LOCAL before initializing it to a particular value. Declaring a variable LOCAL doesn't create it, but it does hide any higher-level variable with the same name. After declaring a variable LOCAL, you can create and initialize it to a value with STORE or =. (The := operator will not work at this point because the variable hasn’t been created yet.) Local variables are erased from memory when the routine that creates them finishes executing.

For more information, see PUBLIC for a table that compares the scope of public, private, local, and static variables.

LOOP

Topic group Related topics

Skips the remaining statements in the current loop, causing another loop iteration to be attempted.

Syntax

LOOP

Description

Conditional statements are often used inside a loop to control which statements are executed in each loop iteration. For example, in a loop that processes the rows in an employee table, you might want to increase the monthly salary of non-managers and the annual bonus for managers, all in the same loop.

There can be many different sets of statements in the loop, each with a different combination of conditions dictating whether they should be executed. Sometimes you can be in the middle of a loop, and none of the remaining statements apply. The condition that determines this may be

nested a few levels deep. While it would be possible to code the rest of the loop with conditional statements to take this condition into account, often it's simpler to use a LOOP statement when this condition is encountered. This causes the remaining statements in the loop to be skipped, and the next iteration of the loop to be attempted.

OTHERWISE

Topic group

Designates a block of code in a DO CASE block to execute if there are no matching CASE blocks.

Description

See DO CASE for details.

PARAMETERS

Topic group Related topics Example

Assigns data passed from a calling routine to private variables.

Syntax

PARAMETERS <parameter list>

<parameter list>

The memory variable names to assign, separated by commas.

Description

There are three ways to access values passed to program or function:

1. Variable names may be declared on the FUNCTION (or PROCEDURE) line in parentheses. These variables are local to that routine.
    - Variable names may be declared in a PARAMETERS statement. These variables are private in scope.
    - The values may be retrieved through the ARGVECTOR( ) function.

Passed values may be assigned to variables only once in a routine. You may either create local variables on the FUNCTION line or use the PARAMETERS statement, and you may only use the PARAMETERS statement once.

The ARGVECTOR( ) function returns copies of the passed values. It has no effect on, nor is it affected by, the other two techniques.

Parameters passed to the main procedure of a *dBASE Plus* application .exe, such as from a DOS command line, will be received as character strings.

For example:

```
someApp abcd efgh
```

In someApp.prg,

```
PARAMETERS var1, var2
```

var1 will be received as, "abcd", and *var2* as, "efgh".

To pass a string containing an embedded space, use quotes around the string. Such as:

```
someApp "abcd efgh" ijk
```

var1 will be received as, "abcd efgh", and *var2* as, "ijk".

In general, local variables are preferred because they cannot be accidentally overwritten by a lower-level routine. Reasons to use PARAMETERS instead include:

- Using values passed to a program file: a program file may contain statements that are not part of a function or class, like the statements in the Header of a WFM file. Because there is no FUNCTION or PROCEDURE line, there is no place to declare local parameters. A PARAMETERS statement must be used instead.
- You specifically want the parameters to be private, so they can for example be modified by a lower-level routine, or be used in a macro substitution.

For more information on the difference between local and private variable scope, see LOCAL.

If you specify more variables in the <parameter list> than values passed to the routine, the extra variables assume a value of *false*. If you specify fewer variables, the extra values do not get assigned.

The PARAMETERS statement should be at or near the top of the routine. This is good programming style; there is no rule requiring this.

Passing mechanisms

There are two ways to pass parameters, by reference or by value. This section uses the term "variable" to refer to both memory variables and properties.

- If you pass variables by reference, the called function has direct access to the variable. Its actions can change (overwrite) the value in that variable. Pass variables by reference if you want the called function to manipulate the values stored in the variables it receives as parameters.
- If you pass variables by value, the called function gets a copy of the value contained in the variable. Its actions can't change the contents of the variable itself. Pass variables by value if you want the called function to use the values in the variables without changing their values—on purpose or by accident—in the calling subroutine.

The following rules apply to parameter passing mechanisms:

- Literal values (like 7) and calculated expression values (like xVar + 3) must be passed by value—there is no reference for the called function to manipulate, nor is there any way to tell that the parameter has been changed.
- Memory variables and properties may be passed by reference or by value. The default is pass-by-reference.
- The scope declaration of a variable (local, private, etc.) does not have any effect on whether the variable is passed by reference or by value. The scope declaration protects the name of the variable. That name is used inside the calling routine; the called function assigns its own name (which is often different but sometimes happens to be the same) to the parameter, making the scope declaration irrelevant.
- To pass a variable or property by value, enclose it in parentheses when you pass it.

Passing objects

Because an object reference is itself a reference, passing one as a parameter is a bit more complicated:

- Passing a variable (or property) that contains an object reference by reference means that you can change the contents of that variable, so that it points to another object, or contains any other value.
- Even if you pass an object reference by value, you can access that object, and change any of its properties. This is because the value of a object reference is still a reference to that object.

Passing *this* and *form*

When passing the special object references *this* and *form* as parameters to the method of another object, they must be enclosed in parentheses to be passed by value. If not, the value of the *this* and *form* parameters take on the corresponding values for the target object, and no longer refer to the calling objects.

Passing fields in XBase DML

With the XBase DML, fields are accessed directly by name (instead of a Field object's *value* property). When used as parameters, they are always passed by value, so the called function can't change their contents.

There are two ways to alter the contents of an XBase field with a function:

Store its contents to a memory variable and call the function with that variable. When control returns to the calling routine, REPLACE the field contents with the memory variable contents.

Design the function to accept a field name. Pass the name of the field, and have the function REPLACE the contents of the named field, using macro substitution to convert the field name to a field reference.

Protecting parameters from change

Because the decision whether to pass by reference or by value is made by the caller, the called function doesn't know whether it's safe to modify the parameter. It's a good idea to copy parameters to work variables and to use those variables instead if their values are going to be changed, unless the intent of the function is specifically to modify the parameters.

parent

Topic group

The immediate container of an object.

Property of

Most data, form, and report objects

Description

Many objects are related in a containership hierarchy. If the container object— referred to as the parent—is destroyed, all the objects it contains—referred to as child objects—are also destroyed. Child objects may be parents themselves and contain other objects. Destroying the highest-level parent destroys all the descendant child objects.

An object's *parent* property refers to its parent object.

For example, a form contains both data objects and visual components. A Query object in a form has the form as its parent. The Query object contains a rowset, which contains an array of fields, which in turn contains Field objects. Each object in the hierarchy has a *parent* property that refers back up the chain, up to the form, which has no parent. A button on the form also has a *parent* property that refers to the form. If the form is destroyed, all of the objects it contains are destroyed.

The *parent* property is often used to refer to sibling objects—other objects that are contained by the parent. For example, one Field object can refer to another by using the *parent* reference to go one level up in the hierarchy, then use the name of the other field to go back down one level to the sibling object.

The *parent* property is read-only.

PCOUNT( )

Topic group

Returns the number of parameters passed to a routine.

Syntax

PCOUNT( )

Description

PCOUNT( ) is identical to ARGCOUNT( ).

PRIVATE

Topic group Related topics

Declares variables that you can use in the routine where they're declared and in all lower-level subroutines.

Syntax

```
PRIVATE <memvar list> |
ALL
  [LIKE <memvar skeleton 1>]
  [EXCEPT <memvar skeleton 2>]
```

<memvar list>

The list of memory variables you want to declare private, separated by commas.

ALL

Makes private all memory variables declared in the subroutine.

LIKE <memvar skeleton 1>

Makes private the memory variables whose names are like the memory variable skeleton you specify for <memvar skeleton 1>. Use characters of the variable names and the wildcards * and ? to create <memvar skeleton 1>.

EXCEPT <memvar skeleton 2>

Makes private all memory variables except those whose names are like the memory variable skeleton you specify for <memvar skeleton 2>. Use characters of the variable names and the wildcards * and ? to create <memvar skeleton 2>. You can use LIKE and EXCEPT in the same statement, for example, PRIVATE ALL LIKE ?_* EXCEPT c_*.

Description

Use PRIVATE in a function to avoid accidentally overwriting a variable with the same name that was declared in a higher-level routine. Normally, variables are visible and changeable in lower-level routines. In effect, PRIVATE hides any existing variable with the same name that was not created in the current routine.It's a good practice to always use LOCAL or PRIVATE. For example, if you write a function that someone else might use, you probably won't know what variables they're using. If you don't use LOCAL or PRIVATE, you might accidentally change the value of one of their variables when they call your function.

Although they have some limitations, local variables are generally preferred over private variables because of their more limited visibilty. You cannot accidentally overwrite them in a lower-level routine, which would happen if you forget to hide a public variable; nor can you inadvertently use a variable created in a higher-level routine, thinking that it's one declared in the current routine, which would happen if you misspell the variable name in the current routine. Also, private variables may be macro-substituted inadvertently with the & operator. For example, if you specify the *text* of a menu item as "&Close" to designate the letter C as the pick character and you happen to have a private variable named close, the variable with be macro-substituted when the menu is created. If the variable was declared local, this wouldn't happen.

You must declare a variable PRIVATE before initializing it to a particular value. Declaring a variable PRIVATE doesn't create it, but it does hide any higher-level variable with the same name. After declaring a variable PRIVATE, you can create and initialize it to a value with STORE or =. (The := operator will not work at this point because the variable hasn't been created yet.) Private variables are erased from memory when the routine that creates them finishes executing.

Unless declared otherwise, variables you initialize in programs are private. If you initialize a variable that has the same name as a variable created in the Command window or declared PUBLIC or PRIVATE in an earlier routine—in other words, a variable that is visible to the current routine—and don't declare the variable PRIVATE first, it is not created as a private variable. Instead, the routine uses and alters the value of the existing variable. Therefore, you should always declare your private variables, even though that is the default.

For more information, see PUBLIC for a table that compares the scope of public, private, local, and static variables.

PROCEDURE

Topic group

Defines a function in a program file including variables to represent parameters passed to the function.

Description

PROCEDURE is identical to FUNCTION. While earlier versions of dBASE differentiated between the two, these differences have been removed. The descriptive terms "function" and "procedure" are used interchangably in *dBASE Plus*. (The term "procedure file" refers to a program file opened with the SET PROCEDURE command, which is not restricted to a file that contains PROCEDUREs only.)

See FUNCTION for details.

PROCREFCOUNT( )

Topic group Related topics

Returns the number of references to a procedure file.

Syntax

PROCREFCOUNT(<procedure file expC>)

<procedure file expC>

The filename or the path and filename of a procedure file.

Description

Use PROCREFCOUNT( ) to find the number of references to a procedure file. PROCREFCOUNT( ) accepts a single parameter which is the name of the procedure file or the full path and name of the procedure file for which you want the count returned.

The returned value is numeric.

Each time a procedure file is loaded it's reference count is incremented by one.

Each time a procedure file is closed it's reference count is decremented by one.

When a procedure file's reference count reaches zero, the procedure file is removed from memory and its contents are no longer accessible.

Use SET PROCEDURE TO <procedure file expC> to load a procedure file.

Use CLOSE PROCEDURE <procedure file expC> to close a procedure file.

PUBLIC

Topic group Related topics

Declares global memory variables.

Syntax

PUBLIC <memvar list>

<memvar list>

The memory variables to make public.

Description

A variable's scope is determined by two factors: its duration and its visibility. A variable's duration determines when the variable will be destroyed, and its visibility determines in which routines the variable can be seen.

Use PUBLIC to declare a memory variable that has an indefinite duration and is available to all routines and to the Command window.

You must declare a variable PUBLIC before initializing it to a particular value. Declaring a variable PUBLIC creates it and initializes it to *false*. Once declared, a public variable will remain in memory until it is explicitly released.

By default, variables you initialize in the Command window are public, and those you initialize in programs without a scope declaration are private. (Variables initialized in the Command window when a program is suspended are private to that program.) The following table compares the characteristics of variables declared PUBLIC, PRIVATE, LOCAL and STATIC in a routine called CreateVar.

	PUBLIC	PRIVATE	LOCAL	STATIC
Created when it is declared and initialized to a value of *false*	Y	N	N	Y
Can be used and changed in CreateVar	Y	Y	Y	Y
Can be used and changed in lower-level routines called by CreateVar	Y	Y	N	N
Can be used and changed in higher-level routines that call CreateVar	Y	N	N	N
Automatically released when CreateVar ends	N	Y	Y	N

Public variables are rarely used in programs. To maintain global values, it's better to create properties of the *_app* object. As properties, they will not conflict with variables that you might have with the same name, and they can communicate with each other more easily.

QUIT

Topic group Example

Closes all open files and terminates *dBASE Plus*.

Syntax

QUIT [WITH <expN>]

WITH <expN>

Passes a return code, <expN>, to the operating system when you exit *dBASE Plus*.

Description

Use QUIT to end your *dBASE Plus* work. It has the same effect as closing the *dBASE Plus* application.

If you include QUIT in a program file, *dBASE Plus* halts the program's execution and exits *dBASE Plus*. To end a program's execution without leaving *dBASE Plus*, use CANCEL or RETURN.

Use QUIT WITH <expN> to pass a return code to Windows or to another application.

REDEFINE

Topic group Related topics

Assigns new values to an object's properties.

Syntax

REDEFINE <class name> <*object name*>
[OF <container object>]
[FROM <row, col> TO <row, col> > | <AT <row, col>]
[PROPERTY <changed property list>]
[CUSTOM <new property list>]

<class name>

The class of the object you want to redefine.

<object name>

The identifier for the object you want to modify. <object name> is either an object reference variable, or a named property of the container if a <containter object> is specified.

OF <container object>

Identifies the object that contains the object you want to redefine.

FROM <row>, <col> TO <row>, <col> | AT <row>, <col>

Specifies the new location and size of the object within its container. FROM and TO specify the upper left and lower right coordinates of the object, respectively. AT specifies the position of the upper left corner.

PROPERTY <changed property list>

Specifies new values you assign to the existing properties of the object.

CUSTOM <new property list>

Specifies new properties you create for the object and the values you assign to them.

Description

Use REDEFINE to assign new values to the properties of an existing object.

While the REDEFINE syntax offers some amenities (like DEFINE), it is not as flexible as assigning values in a WITH block. In particular, with REDEFINE you cannot assign values to the elements of properties that are arrays.

REFCOUNT( )

Topic group Related topics Example

Returns the number of references to an object.

Syntax

REFCOUNT(<oRef>)

<oRef>

Object reference to any valid object

Description

Use REFCOUNT( ) to find the number of references to an object. REFCOUNT( ) accepts a single parameter which is the object reference for which you want the count returned. The returned value is numeric.

RELEASE

Topic group Related topics

Deletes specified memory variables.

Syntax

```
RELEASE <memvar list> |
ALL
  [LIKE <memvar skeleton 1>]
  [EXCEPT <memvar skeleton 2>]
```

<memvar list>

The specific memory variables to release from memory, separated by commas.

ALL

Removes all variables in memory (except system memory variables).

LIKE <memvar skeleton 1>

Removes from memory all memory variables whose names are like the memory variable skeleton you specify for <memvar skeleton 1>. Use characters of the variable names and the wildcards * and ? to create <memvar skeleton 1>.

EXCEPT <memvar skeleton 2>

Removes from memory all memory variables except those whose names are like the memory variable skeleton you specify for <memvar skeleton 2>. Use characters of the variable names and the wildcards * and ? to create <memvar skeleton 2>. You can use LIKE and EXCEPT in the same statement, for example, RELEASE ALL LIKE ?_* EXCEPT c_*.

Description

Use RELEASE to clear memory variables. To remove large groups of variables, use the option ALL [LIKE <memvar skeleton 1>] [EXCEPT <memvar skeleton 2>].

If you issue RELEASE ALL [LIKE <memvar skeleton 1>] [EXCEPT <memvar skeleton 2>] in a program or function, *dBASE Plus* releases only the local and private variables declared in that routine. It doesn't release public or static variables, or variables declared in higher-level routines.

To release a variable by name, that variable must be in scope. For example, you may release a private variable declared in a higher-level routine by name, because the private variable is still visible; but you cannot release a local variable the same way because the local variable is not visible outside its routine.

Note

RELEASE does not explicitly release objects. However, if the only reference to an object is in a memory variable, releasing the variable with RELEASE will in turn release the object. In contrast, RELEASE OBJECT will explicitly release an object, but it does not release any variables that used to point to that object.

When control returns from a subroutine to its calling routine, *dBASE Plus* clears from memory all variables initialized in the subroutine that weren't declared PUBLIC or STATIC. Thus, you don't have to release a routine's local or private variables explicitly with RELEASE before the routine terminates.

RELEASE OBJECT

Topic group Related topics

Explicitly releases an object from memory.

Syntax

RELEASE OBJECT <oRef>

<oRef>

An object reference to the object you want to release.

Description

RELEASE OBJECT functions identically to the *release*( ) method. See *release( )* for details.

Because *release*( ) is a method, its use is preferred, especially when called from a method. But *release*( ) is not a method in all classes. Use RELEASE OBJECT when the object does not have a *release*( ) method, or to release an object regardless of its class.

If <oRef> is a variable, RELEASE OBJECT does not release that variable, or any other variables that point to the just-released object. Testing these variables with EMPTY( ) will return *true* once the object has been released.

RESTORE

Topic group Related topics

Copies the memory variables stored in the specified disk file to active memory.

Syntax

RESTORE FROM <filename> | ? | <filename skeleton>
[ADDITIVE]

<filename> | ? | <filename skeleton>

The file of memory variables to restore. RESTORE FROM ? and RESTORE FROM <filename skeleton> display a dialog box, from which you can select a file. If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, *dBASE Plus* assumes MEM.

ADDITIVE

Preserves existing memory variables when RESTORE is executed.

Description

Use RESTORE with SAVE to retrieve and store important memory variables. All local and private variables are cleared at the end of execution of the routine that created them, while all public and static variables are cleared when you exit *dBASE Plus*. To preserve these values for future use, store them in a memory file by using SAVE. You can then retrieve these values later by using RESTORE.

SAVE saves simple variables only—those containing numeric, string, logical, or null values—and objects of class Array. It ignores all other object reference variables. Therefore you can neither SAVE nor RESTORE objects (other than arrays).

Without the ADDITIVE option, RESTORE clears all existing user memory variables before returning to active memory the variables stored in a memory file. Use ADDITIVE when you want to restore a set of variables while retaining those already in memory.

Note If you use ADDITIVE, and a restored variable has the same name as an existing variable, the restored variable will replace the existing one.

If you issue RESTORE in the Command window, *dBASE Plus* makes all restored variables public. When *dBASE Plus* encounters RESTORE in a program file, it makes all restored variables private to the currently executing function.

RETURN

Topic group Related topics

Ends execution of a program or function, returning control to the calling routine—program or function—or to the Command window.

Syntax

RETURN [<return exp>]

<return exp>

The value a function returns to the calling routine or the Command window.

Description

Programs and functions return to their callers when there are no more statements to execute. When ended this way, they do not return a value.

Use RETURN in a program or function to return a value, or to return before the end of the program or function.

If the RETURN is inside a TRY block, the corresponding FINALLY block, if any, is executed before returning. If there is a RETURN inside that FINALLY block, whatever it returns is returned instead.

SAVE

Topic group Related topics

Stores memory variables to a file on disk.

Syntax

SAVE TO <filename> | ? | <filename skeleton>
[ALL]
[LIKE <memvar skeleton 1>]
[EXCEPT <memvar skeleton 2>]

TO <filename> | ? | <filename skeleton>

Directs the memory variable output to be saved to the target file <filename>. By default, *dBASE Plus* assigns a MEM extension to <filename> and saves the file in the current directory. The ? and <filename skeleton> options display a dialog box in which you specify the name of the target file and the directory to save it in.

ALL

Stores all memory variables to the memory file. If you issue SAVE TO <filename> with no options, *dBASE Plus* also saves all memory variables to the memory file.

LIKE <memvar skeleton 1>

Stores in the target file the memory variables whose names are like the memory variable skeleton you specify for <memvar skeleton 1>. Use characters of the variable names and the wildcards * and ? to create <memvar skeleton 1>.

EXCEPT <memvar skeleton 2>]

Stores in the target file all memory variables except those whose names are like the memory variable skeleton you specify for <memvar skeleton 2>. Use characters of the variable names and the wildcards * and ? to create <memvar skeleton 2>.

Description

Use SAVE with RESTORE to store and retrieve important memory variables. Local and private variables are cleared at the end of the routine that created them, while public and static variables are cleared when you exit *dBASE Plus*. To preserve these values for future use, store them in a memory file with SAVE. Use RESTORE to retrieve them.

If SET SAFETY is ON and a file exists with the same name as the target file, *dBASE Plus* displays a dialog box asking if you want to overwrite the file. If SET SAFETY is OFF, any existing file with the same name is overwritten without warning.

Note SAVE saves simple variables only—those containing numeric, string, logical, or null values—and objects of class Array. It ignores all other object reference variables. Therefore you can neither SAVE nor RESTORE objects (other than arrays). SAVE also does not save function pointer, bookmark, or system memory variables.

SET LIBRARY

Topic group Related topics

Opens a *dBASE Plus* program file as the library file, making its functions, classes, and methods available for execution.

Syntax

SET LIBRARY TO [<filename> | ? | <filename skeleton>]

<filename> | ? | <filename skeleton>

The program file to open. The ? and <filename skeleton> options display a dialog box from which you can select a file. If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, *dBASE Plus* assumes .PRO (a compiled object file). If *dBASE*

Plus can't find a .PRO file, it looks for a .PRG file (a source file). If *dBASE Plus* finds a .PRG file, it compiles it.

Description

SET LIBRARY is similar to SET PROCEDURE. Both commands open a program file, allowing access to the functions, classes, and methods the file contains. The difference is that while SET PROCEDURE can add a program file to a list of procedure files, there can be only one library file open at any time. The library file cannot be closed with the SET PROCEDURE command.

Otherwise, the library file is treated like a procedure file. The library and procedure files are searched in the order they were opened. You may want to designate a stable program file with core functionality as the library file, and all other program files as procedure files.

Issue SET LIBRARY TO without a file name to close the open library file.

SET PROCEDURE

Topic group Related topics

Opens a *dBASE Plus* program file as a procedure file, making its functions, classes, and methods available for execution.

Syntax

SET PROCEDURE TO
[<filename> | ? | <filename skeleton>] [ADDITIVE][PERSISTENT]

<filename> | ? | <filename skeleton>

The procedure file to open. The ? and <filename skeleton> options display a dialog box, from which you can select a file. If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, *dBASE Plus* assumes .PRO (a compiled object file). If *dBASE Plus* can't find a .PRO file, it looks for a .PRG file (a source file). If *dBASE Plus* finds a .PRG file, it compiles it.

ADDITIVE

Prior to dBASE Plus version 2.50

Opens the procedure file(s) without closing any you've opened with previous SET PROCEDURE statements. SET PROCEDURE TO < filename> (without the ADDITIVE option) closes all procedure files you've opened with previous SET PROCEDURE statements other than those tagged PERSISTENT.

Starting with dBASE Plus version 2.50

SET PROCEDURE TO <filename> acts as if ADDITIVE was included. In other words, SET PROCEDURE TO <filename> (without specifying ADDITIVE) will NOT close any open procedure files

PERSISTENT

Opens the procedure file with the PERSISTENT designation and, unless it is specifically referenced in the CLOSE PROCEDURE<filename list>, prevents it from being closed by any means other than CLOSE PROCEDURE PERSISTENT, CLOSE ALL PERSISTENT, or by compiling the file with COMPILE <filename>.

All SET PROCEDURE TO statements, streamed in the form class definition, will have a PERSISTENT designation when the form's *persistent* property is set to true in The Form Designer.

Description

To execute a function or method, that function must be loaded in memory. To be more precise, a simple pointer to that function must be in memory. The contents of the function itself are not necessarily in memory at any given time; if not, the contents get loaded into memory automatically when the function is executed. But if that function's pointer is in memory, it is considered to be loaded.

Whenever you execute a program file with DO (or with the call operator), it is loaded implicitly; pointers to all of the functions, classes, and methods in that file are loaded into memory. Therefore, code in a program file may always call any other functions or methods in the same file.

To access functions, classes, and methods in other program files, load the program file with SET PROCEDURE first. Its function pointers stay in memory until the program file is unloaded with CLOSE PROCEDURE or SET PROCEDURE TO (with no options).

dBASE Plus uses a reference count system to manage program files in memory. Each loaded program file has a counter for the number of times it has been loaded, either explicitly with SET PROCEDURE or implicitly. As long as the count is more than zero, the file stays loaded. Calling CLOSE PROCEDURE reduces the count by one. Therefore, if you issue SET PROCEDURE twice, you need to issue CLOSE PROCEDURE twice to close the program file.

A program file's load count has no impact on memory; it is simply a counter. Loading a program file 10 times uses the same amount of memory as loading it once.

Whenever a function is called, *dBASE Plus* looks for the routine in specific places in a specific order. After searching the program files in the call chain, *dBASE Plus* looks in files opened with SET PROCEDURE. See the DO command for an explanation of the search path and order.

To make the file containing the currently executing routine a procedure file—for example, after creating an object, to make the object's methods which are defined in the same file available to it—execute the following statement:

```
set procedure to program(1) additive
```

Some operations, such as assigning a menuFile to a form or opening a form defined in a WFM file, automatically open the associated file as a procedure file, and that statement is not necessary.

If you issue SET PROCEDURE TO with no options, *dBASE Plus* closes all procedure files opened with SET PROCEDURE other than those tagged PERSISTENT. If you want to close only specific procedure files, use CLOSE PROCEDURE. The maximum number of open procedure files depends on available memory.

Note A common mistake is to forget the ADDITIVE clause when opening a procedure file. This will close all other open procedure files not tagged PERSISTENT.

When *dBASE Plus* executes or loads a program file, it will automatically compile the program file into object code when either:

There is no object code file, or

SET DEVELOPMENT is ON, and program file is newer than the object code file (the source code file's last update date and time is later than the object code file's)

If a file is opened as a procedure file and the file is changed in the Source editor, the file is automatically recompiled so that the changed code takes effect immediately.

Use TYPE( ) to detect whether a function, class, or method is loaded into memory. If so, TYPE( ) will return "FP" (for function pointer), as shown in the following IF statements:

```
if type( "myfunc" ) # "FP" // Function name
if type( "myclass::myclass" ) # "FP" // Class constructor name
```

```
if type( "myclass::mymethod" ) # "FP" // Method name
```

SET( )

Topic group Related topics Example

Returns the current setting of a SET command or function key.

Syntax

SET(<expC> [,<expN>])

<expC>

A character expression that is the SET command or function key whose setting value to return.

<expN>

The nth such setting to return.

Description

Use SET( ) to get a SET or function key setting so that you can change it or save it. For example, you can issue SET( ) at the beginning of a routine to get current settings. You can then save these settings in memory variables, change the settings, and restore the original settings from the memory variables at the end of the routine.

When *dBASE Plus* supports a SET and a SET...TO command that use the same keyword, SET( ) returns the ON|OFF setting and SETTO( ) returns the SET...TO setting. For example, you can issue SET FIELDS ON, SET FIELDS OFF, or SET FIELDS TO <field list>. SET("FIELDS") returns "ON" or "OFF" and SETTO("FIELDS") returns the field list as a character expression.

If *dBASE Plus* supports a SET...TO command but not a corresponding SET command, SET( ) and SETTO( ) both return the SET...TO value. For example, SET("BLOCKSIZE") and SETTO("BLOCKSIZE") both return the same value.

When <expC> is a function key name, such as "F4", SET( ) returns the function key setting. To return the value of a Ctrl+function key setting, add 10 to the function key number; to return the value of a Shift+function key setting, add 20 to the function key number. That is, to return the value of Ctrl+F4, use SET("F14"), and to return the value of Shift+F4, use SET("F24").

If a procedure file is open, SET("PROCEDURE") returns the name of the procedure file. If more than one procedure file is open, SET("PROCEDURE") returns the name of the first one loaded. To return the name of another open procedure file, enter a number as the second argument; for example, SET("PROCEDURE",2) returns the name of the second procedure file that was loaded. If no procedure files are open, SET("PROCEDURE") returns an empty string ("").

The command you specify for <expC> can be abbreviated to four letters in most cases, following the same rules as those for abbreviating keywords. For example, SET("DECI") and SET("DECIMALS") have the same meaning. The <expC> argument is not case-sensitive.

SETTO( )

Topic group Related topics

Returns the current setting of a SET...TO command or function key.

Syntax

SETTO(<expC> [,<expN>])

<expC>

A character expression that is the SET...TO command whose setting value to return.

<expN>

The nth such setting to return.

Description

Use SETTO( ) to get a SET or function key setting so that you can change it or save it. For example, you can issue SETTO( ) at the beginning of a routine to get current settings. You can then save these settings in memory variables, change the settings, and restore the original settings from the memory variables at the end of the routine.

When *dBASE Plus* supports a SET and a SET...TO command that use the same keyword, SET( ) returns the SET setting and SETTO( ) returns the SET...TO setting. For example, you can issue SET FIELDS ON, SET FIELDS OFF, or SET FIELDS TO <field list>. SET("FIELDS") returns the ON or OFF setting and SETTO("FIELDS") returns the field list as a character expression.

SETTO( ) is almost identical to SET( ). For more information, see SET( ).

STATIC

Topic group Related topics Example

Declares memory variables that are local in visibility but public in duration.

Syntax

STATIC <variable 1> [= <value 1>] [,<variable 2> [= <value>] ...]

<variable>

The variable to declare static.

<value>

The value to assign to the variable.

Description

Use STATIC to declare memory variables that are visible only to the routine where they’re declared but are not automatically cleared when the routine ends. Static variables are different from other scopes of memory variables in two important ways:

- You can declare and assign a value to a static variable in a single statement, referred to as an in-line assignment.
- Static variables initialized in a single statement are assigned the initialization value whenever the variable is undefined, including the first time the routine is executed and after the variable is cleared.

You must declare a variable STATIC before initializing it to a particular value. Declaring a variable STATIC without an in-line assignment creates it and initializes it to *false*. Once declared, a static variable will remain in memory until it is explicitly released (usually with CLEAR MEMORY).

Because static variables are not released when the routine in which they are created ends, you can use them to retain values for subsequent times that routine runs. To do this, use an in-line assignment. The first time *dBASE Plus* encounters the STATIC declaration, the variable is initialized to the in-line value. If the subroutine is run again, the variable is not reinitialized; instead, it retains whatever value it had when the routine last ended.

Because dBL is a dynamic object-oriented language, you usually assign new properties to an object to retain values between method calls. For example, if you're calculating a running total in a report, you can create a property of the Report or Group object to store that number.

Static variables are only useful for truly generic functions that are not associated with objects, functions that might be called from different objects that need to share a persistent value, or for values that are maintained by a class—not each object. In this last case, the variables are referred to as static class variables.

For more information, see PUBLIC for a table that compares the scope of public, private, local, and static variables.

STORE

Topic group Related topics

Stores an expression to specified memory variables or properties.

Syntax

STORE <exp> TO <memvar list>

<exp>

The expression to store.

TO <memvar list>

The list of memory variables and/or properties to store <exp>, separated by commas.

Description

Use STORE to store any valid expression to a one or more variables or properties.

Common style dictates the use of STORE only when storing a single value to multiple locations. When storing to a single variable or property, an assignment operator, either = or :=, is preferred.

To specify the scope of a variable, use LOCAL, PRIVATE, PUBLIC, or STATIC before assigning a value to the variable.

THROW

Topic group Related topics Example

Generates an exception.

Syntax

THROW <exception oRef>

<exception oRef>

A reference to the Exception object you want to pass to the CATCH handler.

Description

Use THROW to manually generate an exception. THROW must pass a reference to an existing Exception object that describes the exception.

TRY

Topic group Related topics Example

A control statement used to handle exceptions and other deviations of program flow.

Syntax

```
TRY
 <statement block 1>
CATCH( <exception type1> <exception oRef1> )
 <statement block 2>
[ CATCH( <exception type2> <exception oRef2> )
 <statement block 3> ]
[ CATCH ... ]
[ FINALLY
 <statement block 4> ]
ENDTRY
```

TRY <statement block 1>

A statement block for which the following CATCH or FINALLY block—or both—will be used if an exception occurs during execution. A TRY block must be followed by either a CATCH block, a FINALLY block, or both.

CATCH <statement block 2>

A statement block that is executed when an exception occurs. The first CATCH or FINALLY is not optional and must be included.

<exception type>

The class name of the exception to look for—usually, Exception.

<exception oRef>

A formal parameter to receive the Exception object passed to the CATCH block.

CATCH...

Catch blocks for other types of exceptions.

FINALLY <statement block 4>

A statement block that is always executed after the TRY block, even if an exception or other deviation of program flow occurs. If there is both a CATCH and a FINALLY, the FINALLY block executes after the CATCH block.

ENDTRY

A required keyword that marks the end of the TRY structure.

Description

An exception is a condition that is either generated by *dBASE Plus*, usually in response to an error, or by the programmer. By default, *dBASE Plus* handles an exception by displaying an error dialog and terminating the currently executing program. You can use FINALLY to make sure some code gets executed even if there is an exception, and CATCH to handle the exception yourself, in the following combinations:

- For a block of code that may generate an exception, place the code inside a TRY block. To prevent the exception from generating a standard error dialog and terminating execution, place exception handling code in a CATCH block after the TRY. If an exception occurs, execution immediately jumps to the CATCH block; no more statements in the TRY block are executed. If no exception occurs, the CATCH block is not executed.
- If there's some code that should always be executed at the end of a process, whether or not the process completes successfully, place that code in a FINALLY block. With TRY and FINALLY but no CATCH, if an exception occurs during the TRY block, execution immediately jumps to the FINALLY block; no more statements in the TRY block are executed. Since there was no CATCH, you would still have an exception, which if not handled by a higher-level CATCH as described later, *dBASE Plus* would handle as usual, after executing the FINALLY block. If no exception occurs, the FINALLY block is executed after the TRY.

If you have all three—TRY, CATCH, and FINALLY—if an exception occurs, execution immediately jumps to the CATCH block; after the CATCH block executes, the FINALLY block is executed. If there is no exception during the TRY, then the CATCH block is skipped, and the FINALLY block is executed.

The code that is covered by TRY doesn't have to be inside the statement block physically; the coverage exists until that entire block of code is executed. For example, you may have a function call inside a TRY block, and if an exception occurs while that function is executing—even if that function is defined in another program file—execution jumps back to the corresponding CATCH or FINALLY.

A TRY block may be followed by multiple CATCH blocks, each with its own <exception type>. When an exception occurs, *dBASE Plus* compares the <exception type> with the *className* property of the Exception object. If they match, that CATCH block is executed and all others are skipped. If the *className* does not match, *dBASE Plus* searches the class hierarchy of that object to find a match. If no match is found, the next CATCH block is tested. Class name matches are not case-sensitive. For example, the DbException class is a subclass of the Exception class. If the blocks are arranged like this:

```
try
// Statements
catch ( DbException e )
// Block 1
catch ( Exception e )
// Block 2
endtry
```

and a DbException occurs, execution goes to Block 1, because that's a match. If an Exception occurs, execution goes to Block 2, because Block 1 doesn't match, but Block 2 does. If the blocks are arranged the other way around, like this:

```
try
// Statements
catch ( Exception e )
// Block 1
catch ( DbException e )
// Block 2
endtry
```

then all exceptions always go to Block 1, because all Exceptions are derived from the Exception class. Therefore, when using multiple CATCH blocks, list the most specific exception classes first.

You can generate exceptions on purpose with the THROW statement to control program flow. For example, if you enter deeply nested control structures or subroutines from a TRY block, you can THROW an exception from anywhere in the nested code. This would cause execution to jump back to the corresponding CATCH or FINALLY, instead of having to exit each control structure or subroutine one-by-one.

You may also nest TRY structures. An exception inside the TRY block causes execution to jump to the corresponding CATCH or FINALLY, but an exception in a CATCH or FINALLY is simply treated as an exception. Also, if you have a TRY and FINALLY but no CATCH, that leaves you with an unhandled exception. If the TRY/CATCH/FINALLY is itself inside a TRY block, then that exception would be handled at that next higher level, as illustrated in the following code skeleton:

```
try
// exception level 1
try
// exception level 2
catch ( Exception e )
// handler for level 2
// but exception level 1
```

```
finally
// level 2
endtry
catch ( Exception e )
// handler for level 1
endtry
```

Note that if an exception occurs in the level 2 CATCH, the level 2 FINALLY is still executed before going to the level 1 CATCH, because a FINALLY block is always executed after a TRY block.

In addition to exceptions, other program flow deviations—specifically EXIT, LOOP, and RETURN—are also caught by TRY. If there is a corresponding FINALLY block, it's executed before control is transferred to the expected destination. (CATCH catches only exceptions.)

TYPE( )

Topic group Related topics Example

Returns a character string that indicates a specified expression's data type.

Syntax

TYPE(<expC>)

<expC>

A character string containing the expression whose type to evaluate and return.

Description

Use TYPE( ) to determine the data type of an expression, including whether a variable is undefined.

TYPE( ) expects a character string containing the expression. This allows you to specify a variable name that may not exist. (If you were to use an actual expression with an undefined variable instead of putting the expression in a string, the expression would cause an error.) The <expC> may be any valid character expression, including a variable or a literal string representing the expression to evaluate.

TYPE( ) returns a character string containing a one- or two- letter code indicating the data type. The following table lists the values TYPE( ) returns.

Expression type	TYPE( ) code
Array object	A
DBF or Paradox binary field (BLOB)	B
Bookmark	BM
Character field or string value, Paradox alphanumeric field	C
Codeblock	CB
Date field or value, Paradox date field	D
DateTime value	DT
Float field, Paradox numeric or currency field	F
Function pointer	FP
OLE (general) field	G
Logical field or value	L
DBF or Paradox memo field	M

DBF numeric field or value	N
Object reference (other than Array)	O
Time value	T
Undefined variable, field, invalid expression, or *null*	U

Note that an object of class Array is a special case. Unlike other objects, its code is "A" (this is for backward compatibility with earlier versions of dBASE).

TYPE( ) cannot "see" variables declared as local or static. If there is a public or private variable hidden by a local or static variable of the same name, then TYPE( ) will return the code for that hidden variable. Otherwise, that variable and any expression using that variable is considered undefined.

Use TYPE( ) to detect whether a function, class, or method is loaded into memory. If so, TYPE( ) will return "FP" (for function pointer), as shown in the following IF statements, which detect if the named function is not loaded (this is done to determine if the specified function needs to be loaded):

```
if type( "myfunc" ) # "FP" // Function name
if type( "myclass::myclass" ) # "FP" // Class constructor name
if type( "myclass::mymethod" ) # "FP" // Method name
```

WITH

Topic group Example

A control statement that causes all the variable and property references within it to first assume that they are properties of the specified object.

Syntax

WITH <oRef>
 <statement block>
ENDWITH

<oRef>

A reference to the default object.

<statement block>

A statement block that assumes that the specified object is the default.

ENDWITH

A required keyword that marks the end of the WITH structure.

Description

Use WITH when working with multiple properties of the same object. Instead of using the object reference and the dot operator every time you refer to a property of that object, you specify the object reference once. Then every time a variable or property name is used, it is first checked to see if that name is a property of the specified object. If it is, then that property is used. If not, then the variable or property name is used as-is.

You cannot take advantage of the WITH syntax to create properties. For example:

```
with someObject
existingProperty = 2
newProperty = existingProperty
endwith
```

Suppose that existingProperty is an existing property of the object, and newProperty is not. In the first statement in the WITH block, the value 2 is assigned to the existing property. Then in the second statement, newProperty is treated like a variable, because it does not exist in the object. The statement creates a variable named newProperty, assigning to it the value of the existingProperty property.

Method name conflicts

You may encounter naming conflicts when calling methods inside a WITH block in two ways:

The name of the method matches the name of a built-in function. The built-in function takes precedence. For example, you create a class with a method center( ) and try to call it within a WITH block:

```
with x
center( )
// other code
endwith
```

The CENTER( ) function would be called. It expects parameters, so you'll get a runtime error. You might check your center( ) method, which has no parameters, and wonder what's going on.

It may be possible to call your method by using the explicit object reference, which is normally redundant in a WITH block, and will not work if the object happens to have a property with the same name as the object reference. For example, you could call your center( ) method like this:

```
with x
x.center( )
// other code
endwith
```

If the object x happens to have a property named x, then you would have to create a temporary duplicate reference that does not have the same name as the any other property of x outside the WITH block first:

```
y = x
with x
y.center( )
// other code
endwith
```

The name of the method matches the first word of a command. For example, if the object f has a method named open( ), the method call with the dot operator would look like:

```
f.open( )
```

Using WITH, it would be:

```
with f
open( )
endwith
```

but that code will not work because the name of the method matches the first word in a dBL command; there are some commands that start with the word OPEN. When the compiler sees the word OPEN, it considers that statement to be a command starting with that keyword, and looks for the rest of the command; for example, OPEN DATABASE. When it doesn't find the rest of the command, it considers the statement to be incorrect and generates a compile-time error.

To call such a method inside a WITH block, you may use an explicit object reference as shown above, or change the statement from a direct method call to an indirect method call—an assignment or through the EMPTY( ) function. Many methods return values. By assigning the return value of the method call to variable, even a dummy variable, you bypass the naming

conflict. For example, with another object that has a copy( ) method (there are several commands that begin with the word COPY):

```
with x
dummy = copy( ) // As long as x does not have property named dummy!
endwith
```

For methods that don't return values, you may use the EMPTY( ) function, which will safely "absorb" the undefined value:

```
with x
empty( copy( ) )
endwith
```

Data Objects

Data objects overview

Topic group

Data objects provide access to database tables and are used to link tables to the user interface.

The Borland Database Engine (BDE) considers the DBF (dBASE/FoxPro) and DB (Paradox) tables types as Standard tables. The BDE can access any Standard table directly through its path and file name, without having to use a BDE alias.

All other table types, including InterBase, Oracle, Microsoft SQL Server, Sybase, Informix, and any ODBC connection, require the creation of a BDE alias through the BDE Administrator. You may also create a BDE alias to access Standard tables. In that case, the alias specifies the directory in which the tables exist; the database consists of the Standard tables in that directory, and you may not open any others from another directory unless you explicitly specify the full path without the alias. *See* class Database.

All tables, whether or not they require a BDE alias, are accessed through SQL and the data objects.

Data objects: Class hierarchy

Topic group

To understand the implications of using a BDE alias, you need to understand the class hierarchy of the data objects.

At the top of the hierarchy is *dBASE Plus* itself. Next is the *Session* class. A session represents a separate user task, and is required primarily for DBF and DB table security. *dBASE Plus* supports up to 2048 simultaneous sessions. When *dBASE Plus* first starts, it already has a default session. Unless your application needs to log in as more than one person simultaneously, there is usually no need to create your own session objects.

Each session contains one or more Database objects. A session always contains a default Database object, one that has no BDE alias and is intended to directly access Standard tables. You must create new Database objects to use tables through a BDE alias. Once you set the BDE alias, activate the Database object, and log in if necessary, you have access to that database's tables. You may also log transactions or buffer updates to each database to allow you to rollback, abandon, or post changes as desired.

Accessing tables

The *Query* object acts primarily as a container for an SQL statement and the set of rows, or *rowset*, that results from it. A rowset represents all or part of a single table or group of related tables. There is only one rowset per query, but you may have more than one query, and therefore more than one rowset, per database. A rowset maintains the current record or row, and therefore contains the typical navigation, buffering, and filtering methods.

The SQL statement may also contain parameters, which are represented in the Query object's *params* array.

Finally, a rowset also contains a *fields* property, which is an array of field objects that contain information about the fields and the values of the fields for the current row. There are events that allow you to morph the values so that the values stored in the table are different than the values displayed. Each field object can also be linked to a visual component through the component's *dataLink* property to form a link between the user interface and the table. When the two objects are linked in this way, they are said to be *dataLink*ed.

Putting the data objects together

If you're using Standard tables only, at the minimum you create a query, which gets assigned to the default database in the default session, set the SQL statement and make the query active. If the query is successful, it generates a rowset, and you can access the data through the *fields* array.

When accessing tables through a BDE alias, you will need to create a new database, create the query, assign the database to the query, then set the SQL and make the query active.

If you use the Form or Report designers, you design these relationships visually and code is generated.

Using stored procedures

The object hierarchy for using stored procedures in an SQL-server database is very similar to the one used for accessing tables. The difference is that a StoredProc object is used instead of a Query object. Above the StoredProc object, the Database and Session objects do the same thing. If the stored procedure returns a rowset, the StoredProc object contains a rowset, just like a Query object.

A StoredProc object also has a *params* array, but instead of simple values to substitute into an SQL statement in a Query object, the *params* array of a StoredProc object contains Parameter objects. Each object describes both the type of parameter—input, output, or result—and the value of that parameter.

Before running the stored procedure, input values are set. After the stored procedure runs, output and result values can be read from the *params* array, or data can be accessed through its rowset.

class Database

Topic group Related topics Example

A session's built-in database or a BDE database alias, which gives access to tables.

Syntax

[<oRef> =] new Database( )

<oRef>

A variable or property—typically of a Form or Report object—in which to store a reference to the newly created Database object.

Properties

The following tables list the properties and methods of the Database class. (No events are associated with this class.) For details on each property, click on the property below.

Property	Default	Description
active	false	Whether the database is open and active or closed
baseClassName	DATABASE	Identifies the object as an instance of the Database class
cacheUpdates	false	Whether to cache changes locally for batch posting later
className	(DATABASE)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
databaseName	Empty string	BDE alias, or empty string for built-in database
driverName	Empty string	Type (table format or server) of database
handle		BDE database handle
isolationLevel	Read committed	Isolation level of transaction
loginDBAlias	Empty string	The currently active database alias, or BDE alias, from which to obtain login credentials (user id and password) to be used in activating an additional connection to a database.
loginString	Empty string	User name and password to automatically try when opening database
name	Empty string	The name of custom object
parent	null	Form, SubForm, Report, or Datamodule
session	Default session	Session to which a database is assigned
share	All	How to share the database connection

Event	Parameters	Description
beforeRelease		Before the object is released from memory.

Method	Parameters	Description
abandonUpdates( )		Discards all cached changes
applyUpdates( )		Attempts to post cached changes
beginTrans( )		Begins transaction; starts logging changes
close( )		Closes the database connection (called implicitly when *active* is set to *false*)
commit( )		Commits changes made during transaction; ends transaction
copyTable( )	<source name expC>, <destination name expC>	Makes a copy of a table in the same database
createIndex( )	<table name expC>, <.dbf index oRef>,	Creates an index in the table
dropIndex( )	<table name expC>, <index name expC>	Deletes index from table
dropTable( )	<table name expC>	Deletes table from database
emptyTable( )	<table name expC>	Deletes all records from a table
executeSQL( )	<expC>	Pass-through SQL statement
getSchema( )	"DATABASES" |"TABLES" |"PROCEDURES" | "VIEWS"	Retrieves information about a database

open( )		Opens the database connection (called implicitly when *active* is set to *true*)
packTable( )	<table name expC>	Removes deleted records from DBF or DB table and reconsolidates disk usage
reindex( )	<table name expC>	Rebuilds indexes for DBF or DB table
renameTable( )	<old name expC>, <new name expC>	Renames table in database
rollback( )		Undoes changes made during transaction; ends transaction
tableExists( )	<table name expC>	Whether or not specified table exists in database or on disk

Description

All sessions, including the default session you get when you start *dBASE Plus*, contain a default database, which can access the Standard table types, DBF (dBASE) and DB (Paradox) tables, without requiring a BDE alias. Whenever you create a Query object, it is initially assigned to the default database in the default session. If you want to use Standard tables in the default session, you don't have to do anything with that Query object's *database* or *session* properties. If you want to use a Standard table in another session, for example to use DBF or DB table security, assign that session to the Query object's *session* property, which causes that session's default database to be assigned to that Query object. Default databases are always active; their *active* property has no effect.

You may also set up a BDE alias to access Standard tables. By referring to your Standard tables through a database alias, you can move the tables to a different drive or directory without having to change any paths in your code. All you would have to do is change the path specification for that alias in the BDE Administrator. When using a BDE alias with Standard tables, you must explicitly give the directory path when opening a table in a different directory. You cannot use relative pathing from the directory specified by the alias. For example, if your alias is set to:

C:\MyTables

and you want to use a table somewhere else on the hard drive, such as:

C:\MyTables\TestDir

you must specify the full path without the alias:

C:\MyTables\TestDir or C:\TestDir

For all non-Standard table types, you will need to set up a BDE alias for the database if you haven't done so already. After creating a new Database object, you may assign it to another session if desired; otherwise it is assigned to the default session. Then you need to do the following:

Assign the BDE alias to the *databaseName* property.

If you need to log in to that database, either set the *loginString* property if you already know the user name and password; or let the login dialog appear.

Set the *active* property to *true*. This attempts to open the named database. If it's successful, you now have access to the tables in the database. Methods associated with a Database object will not function properly when the database is not active.

Each database, including any default databases, is able to independently support either transaction logging or cached updates. Transaction logging allows changes to be made to tables as usual, but keeps track of those changes. Those changes can then be undone through a *rollback*( ), or OK'd with a *commit*( ). In contrast, cached updates are not written to the table as they happen, but are cached locally instead. You can then either abandon all the updates or attempt to apply them as a group. If any of the changes fail to post—for a variety of reasons, like

locked records or hardware failures—any changes that did take are immediately undone, and the updates remain cached. You can then attempt to solve the problem and reapply the update, or abandon the changes. You may also want to use cached updates to reduce network traffic.

Each non-Standard database is responsible for its own transaction processing, up to whatever isolation level it supports. For Standard tables opened through the default database, if you want simultaneous multiple transactions, you need to create multiple sessions, because each database object can support only one active transaction or update cache, and there is only one default database per session.

All Database objects opened by the Navigator are listed in the *databases* array property of the _app object. The default database of the default session is _app.*databases*[1].

A Database object also encapsulates a number of table maintenance methods. These methods occur in the context of the specified Database object. For example, the *copyTable*( ) method makes a copy of a table in the same database. To use these methods on Standard tables, call the methods through the default database of the default session; for example,

```
_app.databases[ 1 ].copyTable( "Stuff", "CopyOfStuff" )
```

class DataModule

Topic group Related topics Example

An empty container in which to store data objects.

Syntax

[<oRef> =] new DataModule( )

<oRef>

A variable or property in which to store a reference to the newly created DataModule object.

Properties

The following table lists the properties of the DataModule class. (No events or methods are associated with this class.)

Properties	Default	Description
baseClassName	DATAMODULE	Identifies the object as an instance of the DataModule class
className	(DATAMODULE)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
name	Empty string	The name of a custom object
parent	null	Container, form or report
rowset		The primary rowset of the data module

Event	Parameter	Description
beforeRelease		Before the object is released.

Description

Use data modules to maintain multiple data objects and the relationships between them. Data modules bear some similarity to forms, except that they contain data objects only. Array,

Session, Database, Query, and StoredProc objects are contained inside a DataModule object. They are represented by source code in files with a .DMD extension. You can create custom data modules (in .CDM files) and subclass them.

The relationships between the objects—in particular any *masterSource*, *masterRowset*, or *masterFields* properties—in addition to other properties and event handlers can be set for all the objects in the data module. A primary rowset is assigned in the data module's *rowset* property, just like in a form. The DataModule object is intended to be a simple container. Other than *rowset*, the only other properties associated with this object are *baseClassName*, *className, name* and *parent.*

class DataModRef

Topic group Related topics Example

A reference to a DataModule object.

Syntax

[<oRef> =] new DataModRef( )

<oRef>

A variable or property—typically of a Form or Report object—in which to store a reference to the newly created DataModRef object.

Properties

The following table lists the properties of the DataModRef class. (No events or methods are associated with this class.) For details on each property, click on the property below.

Property	Default	Description
active	false	Whether the referenced dataModule is active
baseClassName	DATAMODREF	Identifies the object as an instance of the DataModule class
className	(DATAMODREF)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
dataModClass		The class name of the dataModule
filename		The name of the file containing the dataModule class
parent	null	Container, form or report
ref	null	A reference to the dataModule object
share	None	How to share the dataModule

Event	Parameter	Description
beforeRelease		Before the object is released.

Description

A DataModRef object is used to access dataModules. The *filename* property is set to the .DMD file that contains the dataModule class definition. The *dataModClass* property is set to the class name of the desired dataModule. Then the *active* property is set to *true* to activate the dataModule.

If the *share* property is All instead of None, any existing instance of the desired dataModule class is used. Otherwise a new instance is created. A reference to the dataModule is assigned to the *ref* property.

When a DataModRef object is activated in the Form designer, the dataModule object's *rowset* property is assigned to the form's *rowset* property. Therefore you can access the form's primary rowset, and all other rowsets relative to it, in the same way whether you're using a dataModule or not. To reference the queries in the dataModule from the form, you have to go through two additional levels of objects. For example, instead of:

```
form.query1.rowset
```

you would have to use:

```
form.dataModRef1.ref.query1.rowset
```

However, if query1.*rowset* was the primary rowset of the dataModule, you would still use:

```
form.rowset
```

anyway, and in query1.*rowset*'s event handlers, you would still use:

```
this.parent.parent.query2.rowset
```

to access query2.rowset whether you're using a dataModule or not, because the two Query objects are in the same relative position in the object containership hierarchy.

class DbError

Topic group Related topics Example

An object that describes a BDE or server error.

Syntax

These objects are created automatically by *dBASE Plus* when a DbException occurs.

Properties

The following table lists the properties of the DbError class. (No events or methods are associated with this class.) For details on each property, click on the property below.

Property	Default	Description
baseClassName	DBERROR	Identifies the object as an instance of the DbError class
className	(DBERROR)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
code		BDE error number
context		Field name, table name, and so on, that caused error
message	Empty string	Text to describe the error
nativeCode		Server error code

Description

When an error using a data object occurs, a DbException is generated. Its *errors* property points to an array of DbError objects.

Each DbError object describes a BDE or SQL server error. If *nativeCode* is zero, the error is a BDE error. If *nativeCode* is non-zero, the error is a server error. The *message* property describes the error.

class DbException

Topic group Related topics Example

An object that describes a data access exception. DbException subclasses the Exception class.

Syntax

These objects are created automatically by *dBASE Plus* when an exception occurs.

Properties

The following table lists the properties of the DbException class. DbException objects also contain those properties inherited from the Exception class. (No events or methods are associated with the DbException class.) For details on each property, click on the property below.

Property	Default	Description
baseClassName	DBEXCEPTION	Identifies the object as an instance of the DbException class
className	(DBEXCEPTION)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
errors		Array of DbError objects

Description

The DbException class is a subclass of the Exception class. It is generated when an error using a data object occurs. In addition to the *dBASE Plus* error code and message, it provides access to BDE and SQL server error codes and messages.

class DbfField

Topic group Related topics

A field from a DBF (dBASE) table. DbfField subclasses the Field class.

Syntax

These objects are created automatically by the rowset.

Properties

The following table lists the properties of the DbfField class. DbfField objects also contain those properties inherited from the Field class. (No events or methods are associated with the DbfField class.) For details on each property, click on the property below.

Property	Default	Description
baseClassName	DBFFIELD	Identifies the object as an instance of the DbfField class
className	(DBFFIELD)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
decimalLength	0	Number of decimal places if the field is a numeric field
default		Default value for field (DBF7 only)
maximum		Maximum allowed value for field (DBF7 only)
minimum		Minimum allowed value for field (DBF7 only)
readOnly	false	Specifies whether the field has read-only access
required	false	Whether the field must be filled in (DBF7 only)

Description

The DbfField class is a subclass of the Field class. It represents a field from a DBF (dBASE) table, and contains properties that are specific to fields of that table type. Otherwise it is considered to be a Field object.

class Field

Topic group Related topics Example

A base class object that represents a field from a table and can be used as a calculated field.

Syntax

[<oRef> =] new Field( )

<oRef>

A variable or property in which to store the reference to the newly created Field object for use as a calculated field.

Properties

The following tables list the properties, events. and methods of the Field class. For details on each property, click on the property below.

Property	Default	Description
baseClassName	FIELD	Identifies the object as an instance of the Field class
className	(FIELD)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
fieldName		Name of the field the Field object represents, or the assigned calculated field name
length		Maximum length
logicalSubType		A database independent name indicating the data subtype of a value stored in a field
logicalType		A database independent name indicating the data type of a value stored in a field
lookupRowset		Reference to lookup table for field
lookupSQL		SQL SELECT statement for field lookup values
parent	null	*fields* array that contains the object
type	Character	The field's data type
value	Empty string	Represents current value of field in row buffer

Event	Parameters	Description
beforeGetValue		When value property is to be read; return value is used as value
canChange	<new value>	When attempting to change value property; return value allows or disallows change
onChange		After value property is successfully changed
onGotValue		After *value* is read

Method	Parameters	Description
copyToFile( )	<filename expC>	Copies data from BLOB field to external file

replaceFromFile( )	<filename expC> [, <append expL>]	Copies data from external file to BLOB field

Description

The Field class acts as the base class for the DbfField (dBASE), PdxField (Paradox), and SqlField (everything else) classes. It contains the properties common to all field types. Each subclass contains the properties specific to that table type. You also create calculated fields with a Field object.

Each rowset has a *fields* property, which points to an array. Each element of that array is an object of one of the subclasses of the Field class, depending on the table type or types contained in the rowset. Each field object corresponds to one of the fields returned by the query or stored procedure that created the rowset.

While the *fieldName*, *length*, and *type* properties describe the field and are the same from row to row, the *value* property is the link to the field's value in the table. The *value* property's value reflects the current value of that field for the current row in the row buffer; assigning a value to the *value* property assigns that value to the row buffer. The buffer is not written to disk unless the rowset's *save*( ) method is explicitly called or there is an implicit save, which is usually caused by navigation in the rowset. You can abandon any changes you make to the row buffer by calling the rowset's *abandon*( ) method.

You may assign a Field object to the *dataLink* property of a control on a form. This makes the control data-aware, and causes it to display the current value of the Field object's *value* property; if changes are made to the control, the new value is written to the Field object's *value* property.

Calculated fields

Use a calculated field to generate a value based on one or more fields, or some other calculation. For example, in a line item table with both the quantity ordered and price per item, you can calculate the total price for that line item. There would be no need to actually store that total in the table, which wastes space.

Because a calculated field is treated like a field in most respects, you can do things like *dataLink* it to a control on a form, show it in a grid, or use it in a report. Because a calculated field does not actually represent a field in a table, writing to its value property directly or changing its value through a *dataLink*ed control never causes a change in a table.

To create a calculated field, create a new Field object and assign it a *fieldName*, then *add*( ) it to the *fields* array of a Rowset object.

Morphed and calculated fields sometimes require display widths that are larger than their field widths. To avoid truncating the display, use a *picture* that represents the field's maximum size.

Note You must assign the *fieldName* before adding the field to the *fields* array.

Because a rowset is not valid until its query opens, you must make the query active before you add the Field object. The query's *onOpen* event, which fires after the query is activated, is a good place to create the calculated field. To set the value of a calculated field, you can do one of two things

Assign a code-reference, either a codeblock or function pointer, to the Field object's *beforeGetValue* event. The return value of the code becomes the Field object's value.

Assign a value to the Field object's *value* property directly as needed, like in the rowset's *onNavigate* event.

class DbfIndex

Topic group Related topics Example

Creates a reference to a DbfIndex object for local tables

Syntax

[<oRef>]=new DbfIndex(<indexname expC>,<expression expC>)

<oRef>

A variable or property in which to store a reference to the newly created DbfIndex object.

<indexname expC>

The name of the index tag for the index

<expression expC>

A dBL expression of up to 220 characters that includes field names, operators, or functions

Properties

The following tables list the properties of the DbfIndex class. No events or methods are associated with this class. For details on each property, click on the property below.

Property	Default	Description
baseClassName	DBFINDEX	Identifies the object as an instance of the DbfIndex class
className	(DBFINDEX)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
descending	False	Creates the index in descending order (Z to A, 9 to 1, later dates to earlier dates). Without DESCENDING, DbfIndex creates an index in ascending order.
expression	Empty string	A dBASE expression of up to 220 characters that includes field names, operators, or functions.
forExpression	Empty string	Limits the records included in the index to those meeting a specific condition.
indexName	Empty string	Specifies the name of the index tag for the index
parent	null	Container, form or report
type	0(MDX)	Identifies the index type. 0=MDX, 1=NDX
unique	False	Prevents multiple records with the same expression value from being included in the index. *dBASE Plus* includes only the first record for each value.

Description

DbfIndex( ) is a subclass of INDEX( ) created specifically for use with DBF tables. If you are using Paradox or SQL tables, see class Index. Use DbfIndex( ) to store a reference in a newly created DbfIndex object. A DbfIndex object requires setting only two properties, indexName and expression. However you may find others, such as descending or unique, particularly helpful. Once you've referenced a DbfIndex object, it's easy to create a new index for your table using the database class method: *createIndex*( ).

indexName

Feel free to name the index whatever you choose. It may be helpful, however, to select an index name that provides some indication of it's function.

Index order (Ascending vs. Descending)

Character keys are ordered in ASCII order (from A to Z, and then a to z); numeric keys are ordered from lowest to highest; and date keys are ordered from earliest to latest (a blank date is higher than all other dates).

class Index

Topic group Related topics Example

An object representing an index from a non-local table

Syntax

[<oRef>]=new Index( )

<oRef>

A variable or property in which to store a reference to the newly created Index object.

Properties

The following tables list the properties of the Index class. No events or methods are associated with this class. For details on each property, click on the property below.

Property	Default	Description
baseClassName	INDEX	Identifies the object as an instance of the Index class
caseSensitive	true	Whether a search string is required to match the case, upper or lower, of a field value.
className	(INDEX)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
descending	false	Creates the index in descending order (Z to A, 9 to 1, later dates to earlier dates). Without DESCENDING, creates an index in ascending order.
fields	Empty string	A list of fields on which the table is indexed
indexName	Empty string	Specifies the name of the index tag for the index
parent	null	Container, form or report
unique	false	Prevents multiple records with the same expression value from being included in the index. *dBASE Plus* includes only the first record for each value.

Description

Use Index( ) to store a reference in a newly created Index object for non-local tables. A subclass of Index, DBFIndex is available when working with local DBF tables (*See* class DBFIndex). An Index object requires setting only two properties, *indexName* and *fields*. As the name implies, *indexName* is the name you'll give the index, and *fields* is a list of fields on which the index is based. Once an Index object has been referenced, use the database class method: *createIndex*( ) to create a new index for your table.

class LockField

Topic group Related topics

A _DBASELOCK field in a DBF table.

Syntax

These objects are created automatically by the rowset.

Properties

The following table lists the properties of the LockField class. (No events or methods are associated with this class.) For details on each property, click on the property below.

Property	Default	Description
baseClassName	LOCKFIELD	Identifies the object as an instance of the LockField class
className	(LOCKFIELD)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
fieldName	_DBASELOCK	Name of the field the LockField object represents (read-only)
lock		Date and time of last row lock
parent	null	*fields* array that contains the object
update		Date and time of last row update
user		Name of user that last locked or updated the row

Description

A LockField object is used to represent the _DBASELOCK field in a DBF table that has been CONVERTed. By examing the properties of a LockField object, you may determine the nature of the last row lock or update.

When a row is locked, either explicitly or automatically, the time, date, and login name of the user placing the lock are stored in the _DBASELOCK field of that row. When a file is locked, this same information is stored in the _DBASELOCK field of the first physical record in the table.

If a DBF table has a _DBASELOCK field, the LockField object is always the last field in the *fields* array, and is referenced by its field name, "_DBASELOCK".

All the properties of a LockField object are read-only.

class Parameter

Topic group Related topics Example

A parameter for a stored procedure.

Syntax

These objects are created automatically by the stored procedure.

Properties

The following table lists the properties of the Parameter class. (No events or methods are associated with this class.) For details on each property, click on the property below.

Property	Default	Description
baseClassName	PARAMETER	Identifies the object as an instance of the Parameter class
className	(PARAMETER)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
type	Input	The parameter type (0=Input, 1=Output, 2=InputOutput, 3=Result)
value		The value of the parameter

Description

Parameter objects represent parameters to stored procedures. Each element of the *params* array of a StoredProc object is a Parameter object. The Parameter objects are automatically created when the *procedureName* property is set, either by getting the parameter names for that

stored procedure from the SQL server or by using parameter names specified directly in the *procedureName* property.

A parameter may be one of four types, as indicated by its *type* property:

1. **Input:** an input value for the stored procedure. The *value* must be set before the stored procedure is called.
    - **Output:** an output value from the stored procedure. The *value* must be set to the correct data type before the stored procedure is called; any dummy value may be used. Calling the stored procedure sets the *value* property to the output value.
    - **InputOutput:** both input and output. The *value* must be set before the stored procedure is called. Calling the stored procedure updates the *value* property with the output value.
    - **Result**: the result value of the stored procedure. In this case, the stored procedure acts like a function, returning a single result value, instead of updating parameters that are passed to it. Otherwise, the *value* is treated like an output value. The name of the Result parameter is always "Result".

A Parameter object may be assigned as the *dataLink* of a component in a form. Changes to the component are reflected in the *value* property of the Parameter object, and updates to the *value* property of the Parameter object are displayed in the component.

class PdxField

Topic group Related topics

A field from a DB (Paradox) table. PdxField subclasses the Field class.

Syntax

These objects are created automatically by the rowset.

Properties

The following table lists the properties of the PdxField class. PdxField objects also contain those properties inherited from the Field class. (No events or methods are associated with the PdxField class.) For details on each property, click on the property below.

Property	Default	Description
baseClassName	PDXFIELD	Identifies the object as an instance of the PdxField class
className	(PDXFIELD)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
default		Default value for field
lookupTable	Empty string	Table to use for lookup value
lookupType	Empty string	Type of lookup
maximum		Maximum allowed value for field
minimum		Minimum allowed value for field
picture	Empty string	Formatting template
required	false	Whether the field must be filled in
readOnly	false	Whether the field has read-only access

Description

This class is called PdxField—not "DbField"—to avoid confusion and simple typographical errors between it and the DbfField class.

The PdxField class is a subclass of the Field class. It represents a field from a DB (Paradox) table, and contains properties that are specific to fields of that table type. Otherwise it is considered to be a Field object.

class Query

Topic group Related topics Example

A representation of an SQL statement that describes a query and contains the resulting rowset.

Syntax

[<oRef> =] new Query( )

<oRef>

A variable or property in which to store a reference to the newly created Query object.

Properties

The following tables list the properties, events, and methods of the Query class. For details on each property, click on the property below:

Property	Default	Description
active	false	Whether the query is open and active or closed
baseClassName	QUERY	Identifies the object as an instance of the Query class
className	(QUERY)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
constrained	false	Whether the WHERE clause of the SQL SELECT statement will be enforced when attempting to update Standard tables
database	null	Database to which the query is assigned
handle		BDE statement handle
masterSource	null	Query that acts as master query and provides parameter values
name	Empty string	The name of custom object
params	AssocArray	Associative array that contains parameter names and values for the SQL statement
parent	null	Container, form or report
requestLive	true	Whether you want a writable rowset
rowset	Object	Reference to the rowset object containing the results of the query
session	null	Session to which the query is assigned
sql	Empty string	SQL statement that describes the query
unidirectional	false	Whether to assume forward-only navigation to increase performance on SQL-based servers
updateWhere	AllFields	Enum to determine which fields to use in constructing the WHERE clause of an SQL UPDATE statement, used for posting changes to SQL-based servers
usePassThrough	false	Controls whether or not a query, with a simple sql select statement (of the form "select * from <table>), is sent directly to the DBMS for execution or is setup to behave like a local database table.

Event	Parameters	Description
beforeRelease		Before the object is released.
canClose		When attempting to close query; return value allows or disallows closure

canOpen	When attempting to open query; return value allows or disallows opening
onClose	After query closes
onOpen	After query first opens

Method	Parameters	Description
execute( )		Executes query (called implicitly when *active* property is set to *true*)
prepare( )		Prepares SQL statement
requery( )		Rebinds and executes SQL statement
unprepare( )		Cleans up when query is deactivated (called implicitly when *active* property is set to *false*)

Description

The Query object is where you specify which fields you want from which rows in which tables and the order in which you want to see them, through an SQL SELECT statement stored in the query's *sql* property. The results are accessed through the query's *rowset* property. To use a stored procedure that results in a rowset, use a StoredProc object instead.

Whenever you create a query object, it is initially assigned to the default database in the default session. If you want to use Standard tables in the default session you don't have to do anything with that query's *database* or *session* properties. If you want to use a Standard table in another session, assign that session to the query's *session* property, which causes that session's default database to be assigned to that query.

For non-Standard tables, you will need to set up a BDE alias for the database if you haven't done so already. After creating a new Database object, you may assign it to another session if desired; otherwise it is assigned to the default session. Once the Database object is active, you can assign it to the query's *database* property. If the database is assigned to another session, you need to assign that session to the query's *session* property first.

After the newly created query is assigned to the desired database, an SQL SELECT statement describing the data you want is assigned to the query's *sql* property.

If the SQL statement contains parameters, the Query object's *params* array is automatically populated with the corresponding elements. The value of each array element must be set before the query is activated. A Query with parameters can be used as a detail query in a master-detail relationship through the *masterSource* property.

Setting the Query object's *active* property to *true* opens the query and executes the SQL statement stored in the *sql* property. If the SQL statement fails, for example the statement is misspelled or the named table is missing, an error is generated and the *active* property remains *false*. If the SQL statement executes but does not generate any rows, the *active* property is *true* and the *endOfSet* property of the query's *rowset* is *true*. Otherwise the *endOfSet* property is *false*, and the rowset contains the resulting rows.

Setting the *active* property to *false* closes the query, writing any buffered changes.

class Rowset

Topic group Related topics Example

The data that results from an SQL statement in a Query object.

Syntax

These objects are created automatically by the query.

Properties

The following tables list the properties, events, and methods of the Rowset class. For details on each property, click on the property below.

Property	Default	Description
autoEdit	true	Whether the rowset automatically switches to Edit mode when a change is made in a *dataLink*ed component.
autoLockChildRows	true	Whether locking a parent row also automatically locks its child rows.
autoNullFields	true	Whether empty fields will assume a null value, or be filled with blanks, zero or, in the case of logical fields, false.
baseClassName	ROWSET	Identifies the object as an instance of the Rowset class
className	(ROWSET)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
codePage	0	Returns a number indicating the current code page associated with a table
endOfSet		Whether the row cursor is at either end of the set
exactMatch	true	Whether rowset searches use a partial string match or an exact string match
fields	Object	Array of field objects in row
filter	Empty string	Filter SQL expression
filterOptions	Match length and case	Enum designating how the filter expression should be applied
handle		BDE cursor handle
indexName	Empty string	Active index tag
languageDriver	Empty string	Returns a character string indicating the name of the language driver currently being used
live	true	Whether the data can be modified
locateOptions	Match length and case	Enum designating how the locate expression should be applied
lockType	0 - Automatic	Enum determinating whether or not explicit locks can be released by a call to rowset.save()
masterChild	Constrained	In a master-detail link, enum specifying whether or not the child table's rowset is constrained.
masterFields	Empty string	Field list for master-detail link
masterRowset	null	Reference to master Rowset object
modified	false	Whether the row has changed
name	Empty string	The name of custom object
navigateByMaster	false	Whether to syncronize movement in a linked-detail rowset to match that of it's master.
navigateMaster	false	Whether to move the row position in a master rowset when a linked-detail rowset reaches endofSet.
notifyControls	true	Whether to automatically update *dataLink*ed controls
parent	null	Query object that contains the Rowset object
state	0	Enum that indicates the rowset's current mode
tableDriver	Empty string	Returns a character string indicating the name of the driver currently being used to access a table

tableLevel	0	Returns an integer indicating the version of the current local table
tableName	Empty string	Returns a character string indicating the name of the table the current rowset is based on
tempTable	false	Returns a logical (True/.T.) when the current table (referenced by *tableName*) is a temporary table

Event	Parameters	Description
beforeRelease		Before the object is released
canAbandon		When *abandon*( ) is called; return value allows or disallows abandoning of row
canAppend		When *beginAppend*( ) is called; return value allows or disallows start of append
canDelete		When *delete*( ) is called; return value allows or disallows deletion
canEdit		When *beginEdit*( ) is called; return value allows or disallows switch to Edit mode
canGetRow		When attempting to read row; return value acts as an additional filter
canNavigate		When attempting row navigation; return value allows or disallows navigation
canSave		When *save*( ) is called; return value allows or disallows saving of row
onAbandon		After successful *abandon*( )
onAppend		After successful *beginAppend*( )
onDelete		After successful *delete*( )
onEdit		After successful *beginEdit*( )
onNavigate	<method expN>, <rows expN>	After rowset navigation
onSave		After successful *save*( )

Method	Parameters	Description
abandon( )		Abandons pending changes to current row
applyFilter( )		Applies filter set during rowset's Filter mode
applyLocate( )	[<locate expC>]	Finds first row that matches specified criteria
atFirst( )		Returns *true* if current row is first row in rowset
atLast( )		Returns *true* if current row is last row in rowset
beginAppend( )		Starts append of new row
beginEdit( )		Puts rowset in Edit mode, allowing changes to fields
beginFilter( )		Puts rowset in Filter mode, allowing entry of filter criteria
beginLocate( )		Puts rowset in Locate mode, allowing entry of search criteria
bookmark( )		Returns bookmark for current row
bookmarksEqual( )	<bookmark 1> [,<bookmark 2>]	Compares two bookmarks or one bookmark with current row to see if they refer to same row
clearFilter( )		Disables filter created by *applyFilter*( ) and clears *filter* property
clearRange( )		Disables constraint created by *setRange*( )

count( )		Returns number of rows in rowset, honoring filters
delete( )		Deletes current row
findKey( )	<key exp>	Finds the row with the exact matching key value
findKeyNearest( )	<key exp>	Finds the row with the nearest matching key value
first( )		Moves row cursor to first row in set
flush( )		Commits the rowset buffer to disk
goto( )	<bookmark>	Moves row cursor to specified row
isRowLocked( )		Determines if the current row, in the current session, is locked
isSetLocked( )		Determines if the current rowset, in the current session, is locked
last( )		Moves row cursor to last row in set
locateNext( )	[<rows expN>]	Finds other rows that match search criteria
lockRow( )		Locks current row
lockSet( )		Locks entire set
next( )	[<rows expN>]	Navigates to adjacent rows
refresh( )		Refreshes entire rowset
refreshControls( )		Refreshes *dataLink*ed controls
refreshRow( )		Refreshes current row only
rowCount ( )		Returns logical row count if known
rowNo( )		Returns logical row number if known
save( )		Saves current row
setRange( )	<key exp> or <startKey exp> | null ,<endKey exp> | null	Constrains the rowset to those rows whose key field values falls within a range
unlock( )		Releases locks set by *lockRow*( ) and *lockSet*( )

Description

A Rowset object represents a set of rows that results from a query. It maintains a cursor that points to one of the rows in the set, which is considered the current row, and a buffer to manage the contents of that row. The row cursor may also point outside the set, either before the first row or after the last row, in which case it is considered to be at the end-of-set. Each row contains fields from one or more tables. These fields are represented by an array of Field objects that is represented by the rowset's *fields* property. For a simple query like the following, which selects all the fields from a single table with no conditions, the rowset represents all the data in the table:

```
select * from CUSTOMER
```

As the cursor moves from row to row, you can access the fields in that row.

A Query object always has a *rowset* property, but that rowset is not open and usable and does not contain any fields until the query has been successfully activated. Setting the Query object's *active* property to *true* opens the query and executes the SQL statement stored in the *sql* property. If the SQL statement fails, for example the statement is misspelled or the named table is missing, an error is generated and the *active* property remains *false*. If the SQL statement executes but does not generate any rows, the *active* property is *true* and the *endOfSet* property of the query's *rowset* is *true*. Otherwise the *endOfSet* property is *false*, and the rowset contains the resulting rows.

Once the rowset has been opened, you can do any of the following:

- Navigate the rowset; that is, move the row cursor
- Filter and search for rows
- Add, modify, and delete rows
- Explicitly lock individual rows or the entire set
- Get information about the rowset, including row cursor's current position

The individual Field objects in a rowset's *fields* array property may be *dataLink*ed to controls on a form. As the row cursor is navigated from row to row, the controls will be updated with the current row's values, unless the rowset's *notifyControls* property is set to *false*. Changing the values shown in the controls will change the *value* property of the *dataLink*ed Field objects. You may also directly modify the *value* property of the Field objects. All of the values are maintained in the row buffer.

Rowset objects support master-detail linking. Navigation and updates in the master rowset change the set of rows in the detail rowset. The detail rowset is controlled by changing the key range of an existing index in the detail rowset. The *masterRowset* and *masterFields* properties are set in the detail rowset. This allows a single master rowset to control any number of detail rowsets.

When a query opens, its rowset is in Browse mode. By default, a rowset's *autoEdit* property is *true*, which means that its fields are changeable through *dataLink*ed controls. Typing a destructive key in a *dataLink*ed control automatically attempts to switch the rowset into Edit mode. By setting *autoEdit* to *false*, the rowset is read-only, and the *beginEdit*( ) method must be called to switch to Edit mode and allow editing. *autoEdit* has no effect on assignments to the *value* of a field; they are always allowed.

The rowset's *modified* property indicates whether any changes have been made to the current row. Changes made to the row buffer are not written until the *save*( ) method is called. However, even after *save*( ) has been called, no attempt is made to save data if the rowset's *modified* property is *false*. This architecture lets you define row-validation code once in the *canSave* event handler that is called whenever it is needed and only when it is needed.

In addition to normal data access through Browse and Edit modes, the rowset supports three other modes: Append, Filter, and Locate, which are initiated by *beginAppend*( ), *beginFilter*( ), and *beginLocate*( ) respectively. At the beginning of all three modes, the row buffer is disassociated from whatever row it was buffering and cleared. This allows the entry of field values typed into *dataLink*ed controls or assigned directly to the *value* property. In Append mode, these new values are saved as a new row if the row buffer is written. In Filter mode, executing an *applyFilter*( ) causes the non-blank field values to be used as criteria for filtering rows, showing only those that match. In Locate mode, calling *applyLocate*( ) causes the non-blank field values to be used as criteria to search for matching rows. In all three modes, using the field values cancels that mode. Also, calling the *abandon*( ) method causes the rowset to revert back to Browse mode without using the values.

You can easily implement filter-by-form and locate-by-form features with the Filter and Locate modes. Instead of using Filter mode, you can assign an SQL expression directly to the rowset's *filter* property. The rowset's *canGetRow* event will filter rows based on any dBL code, not just an SQL expression, and can be used instead of or in addition to Filter mode and the *filter* property. You can also use *applyLocate*( ) without starting Locate mode first by passing an SQL expression to find the first row for which the expression is *true*.

Any row-selection criteria—from the WHERE clause of the query's SQL SELECT statement, the key range enforced by a master-detail link, or a filter—is actively enforced. *applyLocate*( ) will not find a row that does not match the criteria. When appending a new row or changing an

existing row, if the fields in the row are written such that the row no longer matches the selection criteria, that row becomes out-of-set, and the row cursor moves to the next row, or to the end-of-set if there are no more matching rows. To see the out-of-set row, you must remove or modify the selection criteria to allow that row.

Row and set locking support varies among different table types. The Standard (DBF and DB) tables fully support locking, as do some SQL servers. For servers that do not support true locks, the Borland Database Engine emulates optimistic locking. Any lock request is assumed to succeed. Later, when the actual attempt to change the data occurs, if the data has changed since the lock attempt, an error occurs.

Any attempt to change the data in a row, like typing a letter in a *dataLink*ed Entryfield control, causes an automatic row lock to be attempted. If that row is already locked, the lock is retried up to the number of times specified by the session's *lockRetryCount* property; if after those attempts the lock is unsuccessful, the change does not take. If the automatic lock is successful, the lock remains until navigation off the locked row occurs or the row is saved or abandoned; then the lock is automatically removed.

class Session

Topic group Related topics Example

An object that manages simultaneous database access.

Syntax

[<oRef> =] new Session( )

<oRef>

A variable or property—typically of a Form or Report object—in which to store a reference to the newly created Session object.

Properties

The following tables list the properties, events, and methods of the Session class. For details on each property, click on the property below.

Property	Default	Description
baseClassName	SESSION	Identifies the object as an instance of the Session class
className	(SESSION)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
handle		BDE session handle
lockRetryCount	0	Number of times to retry a failed lock attempt
lockRetryInterval	0	Number of seconds to wait between each lock attempt
name	Empty string	The name of custom object. Read-only.
parent	null	Container form or report

Event	Parameters	Description
beforeRelease		Before the object is released
onProgress	<percent expN>, <message expC>	Periodically during data processing operations

Method	Parameters	Description

access( )		Returns the user's access level for the session
addAlias()	<cAlias>, <cDriver>, <cOptions>	adds a User BDE Alias to the current BDE session
addPassword( )	<password expC>	Adds a password to the password table for access to encrypted DB (Paradox) tables
deleteAlias()	<cAlias>	Deletes a User BDE Alias from the current BDE session
login( )	<group expC>, <user expC>, <password expC>	Logs the specified user into the session to access encrypted DBF (dBASE) tables
user( )		Returns the user's login name for the session

Description

A session represents a separate user task, and is required primarily for DBF and DB table security. *dBASE Plus* supports up to 2048 simultaneous sessions. When *dBASE Plus* first starts, it already has a default session.

DBF and DB table security is session-based. (SQL-table security is database-based.) To enable the Session object's security features, the database it is assigned to must be active. When you create a new Session object, it copies the security settings of the default session. Therefore, if you have a user log in when *dBASE Plus* starts, all the new sessions you create to handle multiple tasks will have the access level.

Unlike the Database and Query objects, a Session object does not have an *active* property. Sessions are always active. To close a session, you must destroy it by releasing all references to it.

class SqlField

Topic group Related topics

A field from an SQL-server-based table. SqlField subclasses the Field class.

Syntax

These objects are created automatically by the rowset.

Properties

The following table lists the properties of the SqlField class. SqlField objects also contain those inherited from the Field class. (No events or methods are associated with the SqlField class.) For details on each property, click on the property below.

Property	Default	Description
baseClassName	SQLFIELD	Identifies the object as an instance of the SqlField class
className	(SQLFIELD)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
precision		The number of digits allowed in an SQL-based field
scale		The number of digits, to the right of the decimal point, that can be stored in a SQL-based field

Description

The SqlField class is a subclass of the Field class. It represents a field from an SQL-server-based table, including any ODBC connection, and contains properties that are specific to fields of that table type. Otherwise it is considered to be a Field object.

class StoredProc

Topic group Related topics Example

A representation of a stored procedure call.

Syntax

[<oRef> =] new StoredProc( )

<oRef>

A variable or property—typically of a Form or Report object—in which to store a reference to the newly created StoredProc object.

Properties

The following tables list the properties, events, and methods of the StoredProc class. For details on each property, click on the property below:

Property	Default	Description
active	false	Whether the stored procedure is open and active or closed
baseClassName	STOREDPROC	Identifies the object as an instance of the StoredProc class
className	(STOREDPROC)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
database	null	Database to which the stored procedure is assigned
handle		BDE statement handle
name	Empty string	The name of custom object
params	AssocArray	Associative array that contains Parameter objects for the stored procedure call
parent	null	Container form or report
procedureName	Empty string	Name of the stored procedure
rowset	Object	Reference to the rowset object containing the results of the stored procedure call
session	null	Session to which the stored procedure is assigned

Event	Parameters	Description
beforeRelease		Before the object is released
canClose		When attempting to close stored procedure; return value allows or disallows closure
canOpen		When attempting to open stored procedure; return value allows or disallows opening
onClose		After stored procedure closes
onOpen		After stored procedure first opens

Method	Parameters	Description
execute( )		Executes stored procedure (called implicitly when *active* property is set to *true*)
prepare( )		Prepares stored procedure call
requery( )		Rebinds and executes stored procedure

unprepare( )	Cleans up when stored procedure is deactivated (called implicitly when *active* property is set to *false*)

Description

Use a StoredProc object to call a stored procedure in a database. Most stored procedures take one or more parameters as input and may return one or more values as output. Parameters are passed to and from the stored procedure through the StoredProc object's *params* property, which points to an associative array of Parameter Objects.

Some stored procedures return a rowset. In that case, the StoredProc object is similar to a Query object; but instead of executing an SQL statement that describes the data to retrieve, you name a stored procedure, pass parameters to it, and execute it. The resulting rowset is accessed through the StoredProc object's *rowset* property, just like in a Query object.

Because stored procedures are SQL-server-based, you must create and activate a Database object and assign that object to the StoredProc object's *database* property. Standard tables do not support stored procedures.

Next, the *procedureName* property must be set to the name of the stored procedure. For most SQL servers, the BDE can get the names and types of the parameters for the stored procedure. On some servers, no information is available; in that case you must include the parameter names in the *procedureName* property as well.

Getting or specifying the names of the parameters automatically creates the corresponding elements in the StoredProc object's *params* array. Each element is a Parameter object. Again, for some servers, information on the parameter types is available. For those servers, the *type* properties are automatically filled in and the *value* properties are initialized. For other servers, you must supply the missing *type* information and initialize the *value* to the correct type.

To call the stored procedure, set its *active* property to *true*. If the stored procedure does not generate a rowset, the *active* property is reset to *false* after the stored procedure executes and returns its results, if any. This facilitates calling the stored procedure again if desired, after reading the results from the *params* array.

If the stored procedure generates a rowset, the *active* property remains true, and the resulting rowset acts just like a rowset generated by a Query object.

You can *dataLink* components in a form to fields in a rowset, or to the Parameter objects in the *params* array.

class TableDef

Topic group Related topics

Creates a reference from which to view the definition of a table.

Syntax

[<oRef>]=new TableDef( )

<oRef>

A variable or property in which to store a reference to the newly created TableDef object.

Properties

The following tables list the properties and methods of the TableDef class. No events are associated with this class. For details on each property, click on the property below.

Property	Default	Description
baseClassName	TABLEDEF	Identifies the object as an instance of the TableDef class
className	(TABLEDEF)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
constraints	AssocArray	An array of row-level constraints associated with the table being defined.
database	Object	A reference to the Database object to which the table being defined is assigned.
fields	Object	A reference to an array that contains the table's Field objects
indexes	Object	A reference to an array that contains the table's Index objects
language	Empty string	The Language Driver currently being used to access the table being defined
parent	null	Container, form or report
primaryKey	Empty string	The key expression of the table's primary index
recordCount	zero	Number of records in the table being defined
tableName	Empty string	The name of the table being defined
tableType	DBASE	The current table type
version	7	The tableLevel version number

Method	Parameters	Description
load( )		Loads the table's definition into memory

Description

A TableDef object allows you to view various aspects of a table's definition. Using the TableDef object does not let you make changes to the table's definition, but instead provides a means to read information about it's index tags, fields, constraints and other elements. For information on making changes to a table's definition, see *Table Designer.*

To view a table's definition you must create an instance of the object, provide the table name and load the definition using the TableDef object's load( ) method.

```
t=new TableDef()
t.tableName="tablename"
t.load()
```

Once the table's definition has been loaded, you can view it's contents through The Inspector:

```
inspect(t)
```

or using dot notation from the Command Window:

```
?t.fields.size
?t.fields[n].fieldname // Where n is a number from 1 to the value
of "t.fields.size"
?t.indexes.size
?t.indexes[n].indexname // Where n is a number from 1 to the value of
"t.indexes.size"
```

class UpdateSet

Topic group Related topics Example

An object that updates one table with data from another.

Syntax

[<oRef> =] new UpdateSet( )

<oRef>

A variable or property in which to store a reference to the newly created UpdateSet object.

Properties

The following tables list the properties and methods of the UpdateSet class. (No events are associated with this class.) For details on each property, click on the property below.

Property	Default	Description
baseClassName	UPDATESET	Identifies the object as an instance of the UpdateSet class
changedTableName		Table to collect copies of original values of changed rows
className	(UPDATESET)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
destination		Rowset object or table name that is updated or created
indexName		Name of index to use
keyViolationTableName		Table to collect rows with duplicate primary keys
parent	null	Container, form or report
problemTableName		Table that collects problem rows
source		Rowset object or table name that contains updates

Method	Parameters	Description
append( )		Adds new rows
appendUpdate( )		Updates existing rows and adds new rows
copy( )		Creates destination table
delete( )		Deletes rows in destination that match rows in source
update( )		Updates existing rows

Description

The UpdateSet object is used to update data from one rowset to another, or to copy or convert data from one format to another, either in the same database or across databases.

To update a DBF table with *appendUpdate*( ), *delete*( ), or *update*( ), the *indexName* property of the UpdateSet object must be set to a valid index. To update a DB table with the same operations, the DB table's primary key is used by default, or you can assign a secondary index to the *indexName* property.

The *source* and *destination* can be either a character string containing the name of a table, or an object reference to a rowset. If the source is a rowset, the data used in the update can be filtered.

For Standard table names, specify the name of the table and the extension (DBF or DB). For all other tables, place the database name (the BDE alias) in colons before the table name; that is, in this form:

```
:alias:table
```

The named database must be open when the UpdateSet( ) method is executed.

abandon( )

Topic group Related topics Example

Abandons any pending changes to the current row.

Syntax

<oRef>.*abandon*( )

<oRef>

The rowset whose current row buffer you want to abandon.

Property of

Rowset

Description

Changes made to a row, either through *dataLink*ed controls or by assigning values to the *value* property of fields, are not written to disk until the rowset's *save*( ) method is explicitly called or there is an implicit save, which is usually caused by navigation in the rowset. You can discard any pending changes to the rowset with the *abandon*( ) method. This is usually done in response to the user's request.

You can check the *modified* property first to see if there have been any changes made to the row. Calling *abandon*( ) when there's nothing to abandon has no ill effects (although the *canAbandon* and *onAbandon* events are still fired).

You may also want to discard unwritten changes when a query is closed, the opposite of the default behavior. If you are relying on the query's event handlers to do this instead of abandoning and closing the query through code, you must call *abandon*( ) during the query's *canClose* event and return *true* from the *canClose* event handler; calling *abandon*( ) during the *onClose* event will have no effect, since the *onClose* event fires after the query has already closed, and any changes have been written.

When using *abandon*( ) to discard changes to an existing row, all fields are returned to their original values and any *dataLink*ed controls are automatically restored. If the row was automatically locked when editing began, it is unlocked.

You may also use *abandon*( ) to discard a new row created by the *beginAppend*( ) method, in which case the new row is discarded, and the row that was current at the time *beginAppend*( ) was called is restored. This is not considered navigation, so the rowset's *onNavigate* does not fire. If you have a *onNavigate* event handler, call it from the *onAbandon* event. *abandon*( ) also cancels a rowset's Filter or Locate mode in the same manner.

The order of events when calling *abandon*( ) is as follows:

1. If the rowset has a *canAbandon* event handler, it is called. If not, it's as if *canAbandon* returns *true*.
   - If the *canAbandon* event handler returns *false*, nothing else happens and *abandon*( ) returns *false*.
   - If the *canAbandon* event handler returns *true*:

   The current row buffer/state is abandoned, restoring the rowset to its previous row/state.

   The *onAbandon* event fires.

   abandon( ) returns *true*.

While *abandon*( ) discards unwritten changes to the current row, there are two mutually exclusive ways of abandoning changes to more than one row in more than one table in a database, which you can use instead of or in addition to single-row buffering. Calling *beginTrans*( ) starts transaction logging which logs all changes and allows you to undo them by calling *rollback*( ) if necessary. The alternative is to set the database's *cacheUpdates* property to

true so that changes are written to a local cache but not written to disk, and then call *abandonUpdates*( ) to discard all the changes if needed.

abandonUpdates( )

Topic group Related topics Example

Abandons all cached updates in the database.

Syntax

<oRef>.*abandonUpdates*( )

<oRef>

The database whose cached changes you want to abandon.

Property of

Database

Description

abandonUpdates( ) discards all changes to a database that have been cached. Unlike *applyUpdates*( ), it cannot fail. See *cacheUpdates* for more information on caching updates.

Changes to the current row that have not been written are still in the row buffer, and have not been cached. To abandon changes made to the row buffer, call the rowset's *abandon*( ) method.

access( )

Topic group Related topics Example

Returns the access level of the current session for DBF table security.

Syntax

<oRef>.*access*( )

<oRef>

The session you want to test.

Property of

Session

Description

DBF table security is session-based. All queries assigned to the same session in their *session* property have the same access level.

access( ) returns a number from 0 to 8. 8 is the lowest level of access, 1 is the highest level of access, and 0 is returned if the session is not using DBF security.

active

Topic group Related topics

Specifies whether an object is open and active or closed.

Property of

Database, DataModRef, Query, StoredProc

Description

When created, a new session's default database is active since it does not require any setup. Other Database, DataModRef, Query, and StoredProc objects do require setup, so their *active* property defaults to *false*. Once they have been set up, set their *active* property to *true* to open the object and make it active.

When a Query or StoredProc object's *active* property is set to *true*, its *canOpen* event is called. If there is no *canOpen* event handler, or the event handler returns *true*, the object is activated. In a Query object, the SQL statement in its *sql* property is executed; in a StoredProc object, the stored procedure named in its *procedureName* property is called. Then the object's *onOpen* event is fired.

To close the object, set its *active* property to *false*. Closing an object closes all objects below it in the class hierarchy. Attempting to close a Query or StoredProc object calls its *canClose* event. If there is no *canClose* event handler, or the event handler returns *true*, the object is closed. Closing a Database object closes all its Query and StoredProc objects. After the objects are closed, all the Query and StoredProc objects' *onClose* events are fired.

Activating and deactivating an object implicitly calls a number of advanced methods. You may override or completely redefine these methods for custom data classes; in typical usage, don't touch them. When you set *active* to *true* (methods associated with a Database object will not function properly when the database is not active*)*, a Database object's *open*( ) method is called; activating a query or stored procedure calls *prepare*( ), then *execute*( ). When you set *active* to *false*, a Database object's *close*( ) method is called; deactivating a query or stored procedure calls its *unprepare*( ) method. These methods are called as part of the activation or deactivation of the object, before the *onOpen* or *onClose* event.

Closing a query or a StoredProc object that generated a rowset attempts to write any changes to its rowset's current row buffer, and to apply all cached updates or commit all logged changes. To circumvent this, you must call the *abandon*( ), *abandonUpdates*( ), and/or *rollback*( ) before the object's *onClose* event—for example, during the *canClose* event or before setting the *active* property to *false*—because *onClose* fires after the object has already closed.

Once an object has been closed, you may change its properties if desired and reopen it by setting its *active* property back to *true*.

addPassword( )

Topic group Related topics Example

Adds a password to the session's password list for DB table security.

Syntax

<oRef>.*addPassword*(<expC>)

<oRef>

The session you want to receive the password.

<expC>

The password string.

Property of

Session

Description

DB table security is based on password lists. If you know a password, you have access to all the files that use that password. There is no matching between a user name and password. The access level for each file may be different for the same password.

Password lists are session-based. Once a password has been added to a session, it will continue to be tried for all encrypted tables. All queries assigned to the same session in their *session* property use the same password list. If you attempt to open an encrypted table and there is no valid password that gives access to that table in the list, you will be prompted for the password. Responding with a password adds it to the list.

The *addPassword*( ) method allows you add passwords directly to the session's password list. You can do this if you want to add a default password, so that users won't be prompted, or if you're writing your own custom login form, and need to add the password to the session.

append( )

Topic group Related topics Example

Adds rows from one rowset or table to another.

Syntax

<oRef>.*append*( )

<oRef>

The UpdateSet object that describes the append.

Property of

UpdateSet

Description

Use *append*( ) to add rows from a source rowset or table to an existing destination rowset or table. If there is no primary key in the destination, the rows from the source are always added. If there is a primary key in the destination, rows with keys that already exist in the destination will be copied to the table specified by the UpdateSet object's *keyViolationTableName* property instead.

To update rows with the same primary key in the destination, use the *appendUpdate*( ) method. To move data to a new table instead of an existing table or rowset, use the *copy*( ) method.

When appending multiple rows, be sure the destination and source table structures are exact matches. If the table structures are not exact matches, *append*( ) will terminate when it encounters the discrepency.

appendUpdate( )

Topic group Related topics

Updates one rowset or table from another by updating existing rows and adding new rows.

Syntax

<oRef>.*appendUpdate*( )

<oRef>

The UpdateSet object that describes the update.

Property of

UpdateSet

Description

Use *appendUpdate*( ) to update a rowset, allowing new rows to be added. You must specify the UpdateSet object's *indexName* property which will be used to match the records. The index must exist for the destination rowset. The original values of all changed records will be copied to the table specified by the updateSet's *changedTableName* property.

To update existing rows only, use the *update*( ) method instead. To always add new rows, use the *append*( ) method.

When updating multiple rows, be sure the destination and source table structures are exact matches. If the table structures are not exact matches, *appendUpdate*( ) will terminate when it encounters the discrepency.

applyFilter( )

Topic group Related topics Example

Applies the filter that was set during a rowset's Filter mode.

Syntax

<oRef>.*applyFilter*( )

<oRef>

The rowset whose filter criteria you want to apply.

Property of

Rowset

Description

Rowset objects support a Filter mode in which values can be assigned to Field objects and then used to filter the rows in a rowset to show only those rows with matching values. *beginFilter*( ) puts the rowset in Filter mode and *applyFilter*( ) applies the filter values. *clearFilter*( ) cancels the filter. Because *dataLink*ed controls on forms write to the *value* properties of Field objects, a call to those three methods are all you need to implement a filter-by-form feature in your application.

When *applyFilter*( ) is called, the row cursor is repositioned to the first matching row in the set, or to the end-of-set if there are no matches. The rowset's *filter* property is updated to contain the resulting SQL expression used for the filter. *applyFilter*( ) returns *true* or *false* to indicate if a match was found.

To filter rows with a condition without using Filter mode, set the rowset's *filter* property directly. See the *filter* property for more information on how filters are applied to data. To filter rows with dBL code instead of or in addition to an SQL expression, use the *canGetRow* event.

applyLocate( )

Topic group Related topics Example

Finds the first row that matches specified criteria.

Syntax

<oRef>.*applyLocate*([<SQL condition expC>])

<oRef>

The rowset you want to search for the specified criteria.

<SQL condition expC>

An SQL condition expression.

Property of

Rowset

Description

Rowset objects support a Locate mode in which values can be assigned to Field objects and then used to find rows in a rowset that contains matching values. *beginLocate*( ) puts the rowset in Locate mode and *applyLocate*( ) finds the first matching row. *locateNext*( ) finds other matching rows. Because *dataLink*ed controls on forms write to the *value* properties of Field objects, a call to those three methods are all you need to implement a search-by-form feature in your application.

applyLocate( ) moves the row cursor to the first row that matches the criteria set during the rowset's Locate mode.

You may also use *applyLocate*( ) without calling *beginLocate*( ) first to put the rowset in Locate mode: call *applyLocate*( ) with a parameter string that contains an SQL condition expression. Doing so finds the first row that matches the condition. (Calling *applyLocate*( ) with a parameter when the rowset is in Locate mode discards any field values entered during Locate mode and uses the specified condition expression only to find a match.)

Calling *applyLocate*( ) with a parameter will attempt an implicit save if the rowset is not in Locate mode and the rowset's *modified* property is *true*. If the implicit save fails, because the *canSave* returns *false* or any other reason, the search is not attempted.

If a search is attempted, *applyLocate*( ) returns *true* or *false* to indicate if a match is found. *onNavigate* always fires after a search attempt, either on the first matching row, or the current row if the search failed.

applyLocate( ) will use available indexes to find a match more quickly. When searching on the current index specified by the rowset's *indexName* property, you may find the *findKey*( ) and *findKeyNearest*( ) methods more convenient and direct.

applyUpdates( )

Topic group Related topics

Attempts to apply all cached updates in the database.

Syntax

<oRef>.*applyUpdates*( )

<oRef>

The database whose cached updates you want to apply.

Property of

Database

Description

The *applyUpdates*( ) method attempts to apply all changes to a database that have been cached and returns *true* or *false* to indicate success or failure. If it succeeds, all cached updates are cleared; if it fails, the updates remain cached. Since the *applyUpdates*( ) method uses a transaction while attempting to apply the changes and you cannot nest transactions in a database, cached updates and transaction logging with *beginTrans*( ) are mutually exclusive. See *cacheUpdates* for more information on caching updates.

Changes to the current row that have not been written are still in the row buffer, and have not been cached. To apply changes made to the row buffer, call the rowset's *save*( ) method before you call *applyUpdates*( ).

atFirst( )

Topic group Related topics Example

Returns *true* if the row cursor is at the first row in the rowset.

Syntax

<oRef>.*atFirst*( )

<oRef>

The rowset whose position you want to check.

Property of

Rowset

Description

Use *atFirst*( ) to determine if the row cursor is at the first row in the rowset. When *atFirst*( ) returns *true*, the row cursor is at the first row. In most cases, *atFirst*( ) is an inexpensive operation. The current row is usually compared with a bookmark of the first row made when the query is first opened. However, *atFirst*( ) may be time-consuming for certain data drivers.

A common use of *atFirst*( ) is to conditionally disable backward navigation controls. If you know you are on the first row, you can't go backward, and you reflect this visually with a disabled control.

The end-of-set is different from the first row, so *endOfSet* cannot be *true* if *atFirst*( ) returns *true*. *endOfSet* is *true* if the row cursor is before the first row in the rowset (or after the last row).

Note

Using the rowset's *navigateByMaster* property to synchronize movement in master-detail rowsets, modifies the behavior of the *atFirst*( ) method. See *navigateByMaster* for more information.

atLast( )

Topic group Related topics Example

Returns *true* if the row cursor is at the last row in the rowset.

Syntax

<oRef>.*atLast*( )

<oRef>

The rowset whose position you want to check.

Property of

Rowset

Description

Use *atLast*( ) to determine if the row cursor is at the last row in the rowset. When *atLast*( ) returns *true*, the row cursor is at the last row. *atLast*( ) may be an expensive operation. For example, if you have not navigated to the last row in a rowset returned by an SQL server, such a navigation would have to be attempted to determine if you are at the last row, which could be time-consuming for large rowsets.

A common use of *atLast*( ) is to conditionally disable forward navigation controls. If you know you are on the last row, you can't go forward, and you reflect this visually with a disabled control.

The end-of-set is different from the last row, so *endOfSet* cannot be *true* if *atLast*( ) returns *true*; *endOfSet* is *true* if the row cursor is after the last row in the rowset (or before the first row).

Note

Using the rowset's *navigateByMaster* property to synchronize movement in master-detail rowsets, modifies the behavior of the *atLast*( ) method. See *navigateByMaster* for more information.

autoEdit

Topic group Related topics

Specifies whether the rowset automatically switches to Edit mode when changes are made through *dataLink*ed components.

Property of

Rowset

Description

When a query (or stored procedure) is activated, its rowset opens in Browse mode. If a rowset's *autoEdit* property is *true* (the default), typing a destructive keystroke in a *dataLink*ed component automatically attempts to switch the rowset into Edit mode by implicitly calling *beginEdit*( ). If you set *autoEdit* to *false*, data displayed in a form is read-only, and you must explicitly call *beginEdit*( ) to switch to Edit mode.

autoEdit has no effect on assignments to the *value* of a field; the first assignment to a row always calls *beginEdit*( ) implicitly to secure a row lock.

autoLockChildRows

Topic group Example

Controls whether or not child rows are automaticallly locked (or unlocked) when a parent row is locked (or unlocked).

Property of

Rowset

Description

When *true* (the default), child rows are automatically locked when a parent row is locked.

When *false*, child rows are not locked when a parent row is locked.

The *autoLockChildRows* property can be used to turn off automatic locking of child rows in a data entry form when locking of the child rows is not needed.

A common use for this would be in a data entry form that uses a datamodule where an indexed parent child link is setup between a parent table and a lookup table.

Other rows in the parent table can be linked to the same lookup table rows.

If two users attempt to edit two different parent rows that are linked to the same lookup table row, the second user to attempt an edit will receive an error that the row is locked.

However, if the parent rowset's *autoLockChildRows* property is set to *false*, then the locking conflict would not occur as only the parent rows will be locked.

autoNullFields

Topic group Related topics

Determines whether empty fields are assigned a NULL value, or when applicable, filled with spaces, zero or "false".

Property

Rowset

Description

When the rowset's *autoNullFields* property is set to true (the default setting), *dBASE Plus* allows an empty field to assume a "null value". Null values are those which are nonexistent or undefined. Null is the absence of a value and, therefore, different from a blank or zero value.

When the rowset's *autoNullFields* property is set to false, numeric fields (long, float, etc.) are assigned a value of zero, logical fields a value of "false", and character fields are filled with spaces.

A null value in a field may simply indicate data has yet to be entered, as in a new row, or the field has been purposely left empty . In certain summary operations, null fields are ignored. For example, if you are averaging a numeric field, rows with a null value in the field would not affect the result as they would if the field were filled with zero, or any other value.

beforeGetValue

Topic group Related topics Example

Event fired when reading a field's *value* property, which returns its apparent value.

Parameters

none

Property of

Field (including DbfField, PdxField, SqlField)

Description

By using a field's *beforeGetValue* event, you can make its *value* property appear to be anything you want. For example, in a table you can store codes, but when looking at the data, you see

descriptions. This effect is called field morphing. The *beforeGetValue* event is also the primary way to set up a calculated field.

A field's *beforeGetValue* event handler must return a value. That value is used as the *value* property. During the *beforeGetValue* event handler, the field's *value* property represents its true value, as stored in the row buffer, which is read from the table.

Be sure to include checks for blank values—which will occur when a *beginAppend*( ) starts—and the end-of-set. Any attempt to access the field values when the rowset is at the end-of-set will cause an error. Return a *null* instead.

beforeGetValue is fired when reading a field's *value* property explicitly and when read to update a *dataLink*ed control. It does not fire when accessed internally for SpeedFilters, index expressions, or master-detail links, or when calling *copyToFile*( ).

To reverse the process, use the field's *canChange* event.

Note

Morphed and calculated fields sometimes require display widths that are larger than their field widths. To avoid truncating the display, use a *picture* that represents the field's maximum size.

beginAppend( )

Topic group Related topics Example

Starts append of a new row.

Syntax

<oRef>.*beginAppend*( )

<oRef>

The rowset you want to put in Append mode.

Property of

Rowset

Description

beginAppend( ) clears the row buffer and puts the rowset in Append mode, allowing the creation of a new row, via data entry through *dataLink*ed controls, by directly assigning values to the *value* property of fields, or a combination of both. The row buffer is not written to disk until the rowset's *save*( ) method is explicitly called or there is an implicit save, which is usually caused by navigation in the rowset. At that point, a save attempt is made only if the rowset's *modified* property is *true*; this is intended to prevent blank rows from being added. Calling *beginAppend*( ) again to add another row will also cause an implicit save first, if the row has been modified.

The integrity of the data in the row, for example making sure that all required fields are filled in, should be checked in the rowset's *canSave* event. The *abandon*( ) method will discard the new row, leaving no trace of the attempt.

The rowset's *canAppend* event is fired when *beginAppend*( ) is called. If there is a *canAppend* event handler, it must return *true* or the *beginAppend*( ) will not proceed.

The *onAppend* event is fired after the row buffer is cleared, allowing you to preset default values for any fields. After you preset values, set the *modified* property to *false*, so that the values in the fields immediately after the *onAppend* event are considered as the baseline for whether the row has been changed and needs to be saved.

The order of events when calling *beginAppend*( ) is as follows:

1. If the rowset has a canAppend event handler, it is called. If not, it's as if canAppend returns true.
    - If the canAppend event handler returns false, nothing else happens and beginAppend( ) returns false.
    - If the canAppend event handler returns true, the rowset's modified property is checked.
    - If modified is true:
        - The rowset's canSave event is fired. If there is no canSave event, it's as if canSave returns true.
        - If canSave returns false, nothing else happens and beginAppend( ) returns false.
        - If canSave returns true, *dBASE Plus* tries to save the row. If the row is not saved, perhaps because it fails some database engine-level validation, a DbException occurs—beginAppend( ) does not return.
        - If the row is saved, the modified property is set to false, and the onSave event is fired.
    - Ater the current row is saved (if necessary):
        - The rowset is switched to Append mode.
        - The onAppend event fires.
        - beginA*ppend*( ) returns *true*.

An exception occurs when calling *beginAppend*( ) if the rowset's *live* property is *false*, or if the user has insufficient rights to add rows.

beginEdit( )

Topic group Related topics

Makes contents of a row editable.

Syntax

<oRef>.*beginEdit*( )

<oRef>

The rowset you want to put in Edit mode.

Property of

Rowset

Description

By default, a rowset's *autoEdit* property is *true*, which means that data is immediately editable. The rowset implicitly calls *beginEdit*( ) when a destructive keystroke is typed in a *dataLink*ed component. But you can more strictly control how editing occurs by setting *autoEdit* to *false* and explicitly calling *beginEdit*( ) as needed.

As usual, the row buffer is not written until the rowset's *save*( ) method is explicitly called or there is an implicit save, which is usually caused by navigation in the rowset. The integrity of the data in the row, for example making sure that there are no invalid entries in any fields, should be checked in the rowset's *canSave* event. The *abandon*( ) method will discard any changes to the row. After saving or abandoning any changes, the rowset goes back to Browse mode.

The rowset's *canEdit* event is fired when *beginEdit*( ) is called. If there is a *canEdit* event handler, it must return *true* or the *beginEdit*( ) will not proceed. The *onEdit* event is fired after switching to Edit mode.

The order of events when calling *beginEdit*( ) is as follows, even if the rowset is already in Edit mode:

1. If the rowset has a *canEdit* event handler, it is called. If not, it's as if *canEdit* returns *true*.
    - If the *canEdit* event handler returns *false*, nothing else happens and *beginEdit*( ) returns *false*.

- If the *canEdit* event handler returns *true*:

The rowset attempts to switch to Edit mode by getting an automatic row lock. If the lock cannot be secured, the mode switch fails and *beginEdit*( ) returns *false*.

If the lock is secured, the *onEdit* event fires.

The *beginEdit*( ) method returns *true*.

An exception occurs if the rowset's *live* property is *false*, or if the user has insufficient rights to edit rows, and they call *beginEdit*( ).

beginFilter( )

Topic group Related topics

Puts a rowset in Filter mode, allowing the entry of filter criteria.

Syntax

<oRef>.*beginFilter*( )

<oRef>

The rowset you want to put in Filter mode.

Property of

Rowset

Description

Rowset objects support a Filter mode in which values can be assigned to Field objects and then used to filter the rows in a rowset to show only those rows with matching values. *beginFilter*( ) puts the rowset in Filter mode and *applyFilter*( ) applies the filter values. *clearFilter*( ) cancels the filter. Because *dataLink*ed controls on forms write to the *value* properties of Field objects, a call to those three methods are all you need to implement a filter-by-form feature in your application.

When *beginFilter*( ) is called, the row buffer is cleared. Values that are set either through *dataLink*ed controls or by assigning values to *value* properties are used for matching. Fields whose *value* property is left blank are not considered. To cancel Filter mode, call the *abandon*( ) method.

If navigation is attempted while in Filter mode, Filter mode is canceled and the navigation occurs, relative to the position of the row cursor at the time *beginFilter*( ) was called.

To filter rows with a condition without using Filter mode, set the rowset's *filter* property. See the *filter* property for more information on how filters are applied to data. To filter rows with dBL code instead of or in addition to Filter mode, use the *canGetRow* event.

beginLocate( )

Topic group Related topics

Puts a rowset in Locate mode, allowing the entry of search criteria.

Syntax

<oRef>.*beginLocate*( )

<oRef>

The rowset you want to put in Locate mode.

Property of

Rowset

Description

Rowset objects support a Locate mode in which values can be assigned to Field objects and then used to find rows in a rowset that contain matching values. *beginLocate*( ) puts the rowset in Locate mode and *applyLocate*( ) finds the first matching row. *locateNext*( ) finds other matching rows. Because *dataLink*ed controls on forms write to the *value* properties of Field objects, a call to those three methods are all you need to implement a search-by-form feature in your application.

When *beginLocate*( ) is called, the row buffer is cleared. Values that are set either through *dataLink*ed controls or by assigning values to *value* properties are used for matching. Fields whose *value* property is left blank are not considered. To cancel Locate mode, call the *abandon*( ) method.

If navigation is attempted while in Locate mode, Locate mode is canceled and the navigation occurs, relative to the position of the row cursor at the time *beginLocate*( ) was called.

beginTrans( )

Topic group Related topics

Begins transaction logging.

Syntax

<oRef>.*beginTrans*( )

<oRef>

The database in which you want to start transaction logging.

Property of

Database

Description

Separate changes that must be applied together are considered to be a transaction. For example, transferring money from one account to another means debiting one account and crediting another. If for whatever reason one of those two changes cannot be done, the whole transaction is considered a failure and any change that was made must be undone.

Transaction logging records all the changes made to all the tables in a database. If no errors are encountered while making the individual changes in the transaction, the transaction log is cleared with the *commit*( ) method and the transaction is done. If an error is encountered, all changes made so far are undone by calling the *rollback*( ) method.

Transaction logging differs from caching updates in that changes are actually written to the disk. This means that others who are accessing the database can see your changes. In contrast, with cached updates your changes are written all at once later, when and if you decide to post the changes. For example, if you’re reserving seats on an airplane, you want to post a reservation as soon as possible. If the customer changes their mind, you can undo the reservation with a rollback. With cached updates, the seat might be taken by someone else between the time the data entry for the reservation begins and the time it is actually posted.

All locks made during a transaction are maintained until the transaction is completed. This ensures that no one else can make any changes until the transaction is committed or abandoned.

For SQL-server databases, the Database object's *isolationLevel* property determines the isolation level of the transaction.

A Database object may have only one transaction active at one time; you cannot nest transactions.

bookmark( )

Topic group Related topics Example

Returns the current position in a rowset.

Syntax

<oRef>.*bookmark*( )

<oRef>

The rowset whose current position you want to return.

Property of

Rowset

Description

A bookmark represents a position in a rowset. *bookmark*( ) returns the current position in the rowset. The bookmark may be stored in a variable or property so that you can go back to that position later with the *goto*( ) method.

A bookmark is guaranteed to be valid only as long as the rowset stays open. The bookmark uses the current index represented by the *indexName* property, if any. The same physical row in the table returns different bookmarks when different indexes are in effect. When you *goto*( ) a bookmark, the index that was in effect when the bookmark was returned is automatically activated.

bookmarksEqual( )

Topic group Related topics

Checks if a given bookmark matches the current row, or if two bookmarks refer to the same row.

Syntax

<oRef>.*bookmarksEqual*(<bookmark1> [, <bookmark2>])

<oRef>

The rowset in which to check the bookmark(s).

<bookmark1>

The bookmark to check against the current row in the rowset, if only one bookmark is specified; or the first of two bookmarks to compare.

<bookmark2>

The second of two bookmarks to compare.

Property of

Rowset

Description

Use *bookmarksEqual*( ) to check a bookmark against the current row, without having to first use *bookmark*( ) to get a bookmark for the current row. If the bookmark refers to the current row, *bookmarksEqual*( ) returns *true*; if not it returns *false*. You may also use *bookmarksEqual*( ) to compare two bookmarks to see if they refer to the same row; the equality operators (= and ==) may also be used to compare two bookmarks.

The bookmark uses the current index represented by the *indexName* property, if any. The same physical row in the table returns different bookmarks when different indexes are in effect. When checking a bookmark against the current row, the rowset must be in the same index order as the bookmark; otherwise *bookmarksEqual*( ) will return *false*. When comparing two bookmarks, they must have been taken when the same index was in effect; if not, they will not match.

cacheUpdates

Topic group Related topics

Whether to cache updates locally instead of writing to disk as they occur.

Property of

Database

Description

Normally, when a row buffer is saved, it is written to disk. By setting the *cacheUpdates* property to *true*, those changes are cached locally instead of being written to disk. One reason to do this is to reduce network traffic. Changes are accumulated and then posted with the *applyUpdates*( ) method, after a certain amount of time or a certain number of changes have been made.

Another reason is to simulate a transaction when you have more than one change in an all-or-nothing situation. For example, if you need to fill a customer order and reduce the stock in inventory, you cannot let one happen and not the other. When the changes are posted with *applyUpdates*( ), they are applied inside a transaction at the database level. Because you cannot nest transactions, you cannot have a transaction with *beginTrans*( ) and use cached updates at the same time. If any of the changes do not post, for example one of the records is locked, all of the changes that did post are undone and *applyUpdates*( ) returns *false* to indicate failure. The cached updates remain cached so that you can retry the posting. If all the changes are posted successfully, *applyUpdates*( ) returns *true*.

Finally, because of the all-or-nothing nature of cached updates, you can use them to allow the user to tentatively make changes that you can simply discard as a group. For example, you could allow a user to modify a lookup table. If the user submits the changes they are applied, but if the user chooses to cancel, any changes made can be discarded by calling the *abandonUpdates*( ) method. Note that with cached updates, the changes aren't actually written until posted. In contrast, transaction logging actually makes the changes as they happen, but allows you to undo them if desired.

canAbandon

Topic group Related topics Example

Event fired when attempt to abandon rowset occurs; return value determines if changes to row are abandoned.

Parameters

none

Property of

Rowset

Description

A rowset may be abandoned explicitly by calling its *abandon*( ) method, or implicitly via the user interface by pressing Esc or choosing Abandon Row from the default Table menu or toolbar while editing table rows. *canAbandon* may be used to verify that the user wants to abandon any changes that they have made. You may check the *modified* property first to see if there are any changes to abandon; if not, there is no need to ask.

The *canAbandon* event handler must return *true* or *false* to indicate whether the changes to the rowset, if any, are abandoned.

canAppend

Topic group Related topics

Event fired when attempting to put rowset in Append mode; return value determines if the mode switch occurs.

Parameters

none

Property of

Rowset

Description

A rowset may be put in Append mode explicitly by calling its *beginAppend*( ) method, or implicitly via the user interface by choosing Append Row from the default Table menu or toolbar while editing table rows. *canAppend* may be used to verify that the user wants to add a new row. You can check the *modified* property first to see if the user has made any changes to the current row; if not, you may not want to ask.

The *canAppend* event handler must return *true* or *false* to indicate whether *beginAppend*( ) proceeds. For information on how *canAppend* interacts with other events and implicit saves, see *beginAppend( )*.

canChange

Topic group Related topics Example

Event fired when a change to the *value* property of a Field object is attempted; return value determines if the change occurs.

Parameters

<new value>

The proposed new value

Property of

Field

Description

Use *canChange* to determine whether changes to individual fields occur. *canChange* fires when something is assigned to the *value* property of a Field object, either directly or through a *dataLink*ed control. The proposed new value is passed as a parameter to the *canChange* event handler. If the *canChange* event handler returns *false*, the Field object's *value* remains unchanged.

While *canChange* provides field-level validation to see whether changes are saved into the row buffer, use *canSave* to provide row-level validation to determine whether the buffer can be saved to disk. You should always do row-level validation no matter whether you do field-level validation or not.

The *canChange* event operates separately from database engine-level validation. Even if *canChange* returns *true*, attempting to write an invalid value to a field, for example exceeding a field's maximum allowed value, will fail and the field's *value* property will remain unchanged.

You can also use *canChange* to reverse the field morphing performed by *beforeGetValue*. Inside the *canChange* event handler, examine the <new value> parameter and assign the value you want to store in the table directly to the *value* property of the Field object. Doing so does not fire *canChange* recursively. Then have the *canChange* event handler return *false* so that the <new value> does not get saved into the row buffer.

canClose

Topic group Related topics

Event fired when there's an attempt to deactivate a query or stored procedure; return value determines if the object is deactivated.

Parameters

none

Property of

Query, StoredProc

Description

If the *active* property of a Query or StoredProc object is set to *false*, that object's *canClose* event fires. If the *canClose* event handler returns *false*, the close attempt fails and the *active* property remains *true*.

A StoredProc object may be deactivated only if it returns a rowset. If it returns values only, the *active* property is automatically reset to *false* after the stored procedure is called; there is nothing to deactivate.

Normally when a Query or StoredProc object closes, it saves any changes in its rowset's row buffer, if any. In attempting to save those changes, the rowset's *canSave* event is also fired, before *canClose*. If *canSave* returns *false*, the row is not saved, and the object is not closed.

If you want to abandon uncommitted changes instead of saving them when closing the object, call the rowset's *abandon*( ) method before closing.

canDelete

Topic group Related topics

Event fired when attempting to delete the current row; return value determines if the row is deleted.

Parameters

none

Property of

Rowset

Description

A row may be deleted explicitly by calling the *delete*( ) method, or implicitly via the user interface by choosing Delete Rows from the default Table menu or toolbar. *canDelete* may be used to make sure that the user wants to delete the current row.

canDelete may also be used to do something with the current row, just before you delete it. In this case, the *canDelete* event handler would always return *true*.

The *canDelete* event handler must return *true* or *false* to indicate whether the row is deleted. For information on how *canDelete* interacts with other events, see *delete( )*.

canEdit

Topic group Related topics

Event fired when attempting to put rowset in Edit mode; return value determines if the mode switch occurs.

Parameters

none

Property of

Rowset

Description

The *beginEdit*( ) method is called (implicitly or explicity) to put the rowset in Edit mode. *canEdit* may be used to verify that the user is allowed to or wants to edit the row.

The *canEdit* event handler must return *true* or *false* to indicate whether the switch to Edit mode proceeds.

canGetRow

Topic group Related topics Example

Event fired when attempting to read a row into the row buffer; return value determines if the row stays in or is filtered out.

Parameters

none

Property of

Rowset

Description

In addition to setting an SQL filter expression in the *filter* property, you can filter out rows through dBL code with *canGetRow*. In a *canGetRow* handler, the rowset acts as if the row is read into the row buffer. You can test the *value* properties of the field objects, or anything else.

If *canGetRow* returns *true*, that row is kept. If it returns *false*, the row is discarded and the next row is tried.

Note that *canGetRow* fires before applying the constrain on a detail table linked through *masterRowset* or *masterSource*. Therefore, when using this type of link, you cannot check for the existence of detail rows (by checking the detail rowset's *endOfSet* property) or get the values of the first matching detail row in the *canGetRow* event handler. To access the matching rows in the linked table during the *canGetRow* event, you must manually apply the constrain (using the *setRange*( ) or *requery*( ) methods) inside the *canGetRow* instead of using the built-in properties. Then you are free to access the detail table as usual.

canNavigate

Topic group Related topics

Event fired when attempting navigation in a rowset; return value determines if row cursor is moved.

Parameters

none

Property of

Rowset

Description

Navigation in a rowset may occur explicitly by calling a navigation method like *next*( ) or *goto*( ), or implicitly via the user interface by choosing a navigation option from the default Table menu or toolbar while viewing a rowset. *canNavigate* may be used to verify that the user wants to leave the current row to go to another. You may check the *modified* property first to see if the user has made any changes to the current row; if not, you may not want to ask.

canNavigate may also be used to do something with the current row, just before you leave it. In this case, the *canNavigate* event handler would always return *true*.

The *canNavigate* event handler must return *true* or *false* to indicate whether the navigation occurs. For information on how *canNavigate* interacts with other events and implicit saves, see *next*( ).

canOpen

Topic group Related topics

Event fired when attempting to open a query or stored procedure; return value determines if object is opened.

Parameters

none

Property of

Query, StoredProc

Description

canOpen fires when a Query or StoredProc object's *active* property is set to *true*.

If an event handler is assigned to the *canOpen* property, the event handler must return *true* or *false* to indicate whether the object is opened and activated.

canOpen may also be used to do something with the query, just before you open it. In this case, the *canOpen* event handler would always return *true*.

canSave

Topic group Related topics Example

Event fired when attempting to save the row buffer; return value determines if the buffer is written.

Parameters

none

Property of

Rowset

Description

The row buffer may be saved explicitly by calling *save*( ) or implicitly, usually by navigating in the rowset. Use *canSave* to verify that the data is good before attempting to write it to the disk.

The *canSave* event handler must return *true* or *false* to indicate whether the row is saved. If the user has changed the current row and attempts to append a new row or navigate, *canAppend* or *canNavigate* fires first. If that event returns *true*, then the *canSave* event fires. If *canSave* returns *false*, the row is not saved, and the attempted action does not occur. If *canSave* returns *true*, then the row is saved and the action occurs. This allows you to put row validation code in the *canSave* event handler that you do not need to duplicate in either *canAppend* or *canNavigate*.

The *canSave* event operates separately from database engine-level validation. Even if *canSave* returns *true*, attempting to write an invalid row, for example one that fails to pass a table contraint, will fail and cause an exception.

changedTableName

Topic group Related topics

Name of the table for which you want to collect copies of original values of rows that were changed.

Property of

UpdateSet

Description

When doing an *update*( ) or *appendUpdate*( ), rows will be changed. The original contents of the rows that are changed are copied to the table specified by the *changedTableName* property. If

the table does not exist, it is created. If it does exist, it is erased first so that it contains only those rows that were changed on the last update.

By making copies of the original values of the rows that are changed, you can undo the changes by doing another *update*( ), using the *changedTableName* table as the *source* table.

clearFilter( )

Topic group Related topics

Clears any active filter on a rowset.

Syntax

<oRef>.*clearFilter*( )

<oRef>

The rowset whose filter to clear.

Property of

Rowset

Description

clearFilter( ) clears the *filter* property and any filter set through the rowset's Filter mode, thereby deactivating any filters. Rows that were hidden by the filter become visible. The row cursor is not moved.

clearRange( )

Topic group Related topics

Clears any active range on a rowset.

Syntax

<oRef>.*clearRange*( )

<oRef>

The rowset whose range to clear.

Property of

Rowset

Description

clearRange( ) clears the range set by the *setRange*( ) method. The row cursor is not moved.

close( )

Topic group Related topics

Closes a database connection.

Syntax

This method is called implicitly by the Database object.

Property of

Database

Description

The *close*( ) method closes the database connection. It is called implicitly when you set the Database object's *active* property to *false*. In typical usage, you do not call this method directly.

Advanced applications may override the definition of this method to perform supplementary actions when closing the database connection. Custom data drivers must define this method to perform the appropriate actions to close their database connection.

codePage

Topic group Related topics

The current code page number asssociated with a table

Property of

Rowset

Description

For characters whose ASCII values are between 128 and 255, a code page number identifies which character set is used. *codePage* will return a non-zero value only when the BDE detects a code page in a table's header. Read-only.

commit( )

Topic group Related topics

Clears the transaction log, committing all logged changes

Syntax

<oRef>.*commit*( )

<oRef>

The database whose changes you want to commit.

Property of

Database

Description

A transaction works by logging all changes. If an error occurs while attempting one of the changes, or the changes need to be undone for some other reason, the transaction is canceled by calling the *rollback*( ) method. Otherwise, *commit*( ) is called to clear the transaction log, thereby indicating that all the changes in the transaction were committed and that the transaction as a whole was posted.

constrained

Topic group Related topics

Specifies whether updates to a rowset will be constrained by the WHERE clause of the query's SQL SELECT command. Applies to Standard tables only.

Property of

Query

Description

When *constrained* is set to *true*, any time a row is saved, if the query's SQL SELECT statement—which was stored in the *sql* property and used to generate the rowset—contains a WHERE clause, the newly saved row is evaluated against the WHERE clause. If the row no longer matches the condition set by the WHERE clause, the row is considered to be out-of-set, and the row cursor moves to the next row in the set, or to the end-of-set if already at the last row.

This property applies only to Standard tables and defaults to *false*, which means that the SQL SELECT statement is used only to generate the rowset, not to actively constrain it. By setting the *constrained* property to *true*, Standard tables behave more like SQL-server based tables, which always constrain rows according to the WHERE clause.

copy( )

Topic group Related topics

Copies a rowset or table to a new table.

Syntax

<oRef>.*copy*( )

<oRef>

The UpdateSet object that describes the copy.

Property of

UpdateSet

Description

Use the UpdateSet's *copy*( ) method copy a rowset to a new table in the same database, or to a new table in a different database.

The *source* and *destination* properties specify what to copy and where to copy it. Because you can use a rowset as a *source*, you can copy only part of a table by selecting only those rows you want to copy for the rowset. When using a table name as a *destination*, that table is created, or overwritten if it already exists. To convert from one table type to another, create a rowset of the desired result type and assign it to the *destination* property.

Note: Existing tables used as a *destination* will be overwritten without warning, regardless of the SET SAFETY setting.

To copy all of the rows from a single table in a database to another new table in the same database, use the Database's *copyTable*( ) method.

copyTable( )

Topic group Related topics

Makes a copy of one table to create another table in the same database.

Syntax

<oRef>.*copyTable*(<source table expC>, <destination table expC>)

<oRef>

The database in which you want to copy the table.

<source table expC>

The name of the table you want to duplicate.

<destination table expC>

The name of the table you want to create.

Property of

Database

Description

copyTable( ) copies all of the rows from a single table in a database to another new table in the same database. The resulting destination table will be the same table type as the source table. Use the UpdateSet's *copy*( ) method for any other type of row copy.

The table to copy should not be open.

To make a copy of a Standard table, you can always use the default database in the default session by referring to it through the *databases* array property of the _app object. For example,

```
_app.databases[ 1 ].copyTable( "Stuff", "CopyOfStuff" )
```

copyToFile( )

Topic group Related topics Example

Copies the contents of a BLOB field to a new file.

Syntax

<oRef>.*copyToFile*(<file name expC>)

<oRef>

The BLOB field to copy.

<file name expC>

The name of the file you want to create.

Property of

Field

Description

copyToFile( ) copies the specified BLOB field (including memo fields) to the named file.

count( )

Topic group Related topics

Returns the number of rows in a rowset, respecting any filter conditions and events.

Syntax

<oRef>.*count*( )

<oRef>

The rowset you want to measure.

Property of

Rowset

Description

count( ) returns the number of rows in the current rowset. For a rowset generated by a simple query like the following, which selects all the fields from a single table with no conditions, *count*( ) returns the number of rows in the table:

```
select * from CUSTOMER
```

You can use *count*( ) while a filter is active—with the *filter* property or the *canGetRow* event—to count the number of rows that match the filter condition. This may be time-consuming with large rowsets.

createIndex( )

Topic group Related topics

The createIndex( ) method creates an index for a specified table.

Syntax

createIndex (<tablename expC>,<oRef>)

<table name expC>

The name of the table on which you want to create the index.

<oRef>

Predefined .dbf index object

Property of

Database

Description

The *createIndex*( ) method creates an index from an instance of a database index object. Before using *createIndex*( ):

Close all active queries.

The .dbf index object's name and expression properties must be defined (see following example), and cannot include calculated fields, UDFs, or, since no queries are active, fields in a lookupRowset.

```
ex. d=new DbfIndex()
  d.indexName="indextagname"
  d.expression="indexexpression" // other properties
  _app.databases[1].createIndex("tablename", d)
```

database

Topic group Related topics

The Database object to which the query, stored procedure or table definition is assigned.

Property of

Query, StoredProc, TableDef

Description

A query or stored procedure must be assigned to the database that provides access to the tables it wants before it is activated. When created, a Query or StoredProc object is assigned to the default database in the default session.

To assign the object to the default database in another session, assign that session to the *session* property. Assigning the *session* property always sets the *database* property to the default database in that session.

To assign the object to another database in another session, assign the object to that session first. This makes the databases in that session available to the object.

databaseName

Topic group Related topics

The BDE alias that the object represents.

Property of

Database

Description

To use a BDE alias, create a Database object and assign the alias to the object's *databaseName* property. Then set the *active* property to *true* to activate the database. While the database is active, you cannot change the *databaseName* property.

The *databaseName* property for a session's default database is always blank.

dataModClass

Topic group Related topics

The class name of the desired data module.

Property of

DataModRef

Description

After setting the *filename* property to the file that contains the data module class definition, set the *dataModClass* property to the name of the desired class.

Note

When declaring a class name, the name may exceed 32 characters, but the rest are ignored. When attempting to use a class, the name should not exceed 32 characters; otherwise the named class may not be found.

decimalLength

Topic group Related topics

The number of decimal places in a DBF (dBASE) numeric or float field.

Property of

DbfField

Description

The DBF (dBASE) table format supports two kinds of fields that store numbers: numeric and float. Both field types have a fixed number of decimal places. The *decimalLength* property represents the number of decimal places for any Field objects that represent a numeric or float field. For other field types, *decimalLength* is zero.

default

Topic group Related topics

The default value for a field.

Property of

DbfField, PdxField

Description

default indicates the default value of the field represented by the field object. When a rowset switches to Append mode to add a new row, the field objects take on their default values.

For date fields, the special value TODAY indicates today's date. For timestamp fields, the special value NOW indicates the current date and time.

delete( ) [Rowset]

Topic group Related topics Example

Deletes the current row.

Syntax

<oRef>.*delete*( )

<oRef>

The rowset whose current row you want to delete.

Property of

Rowset

Description

delete( ) deletes the current row in the rowset. When *delete*( ) is called, the *canDelete* event is fired. If there is no *canDelete* event handler, or the event handler returns *true*, the current row is deleted, the *onDelete* event fires, and the row cursor moves to the next row, or to the end-of-set if the last row was the one deleted. This movement is not considered navigation, so the rowset's *onNavigate* does not fire. If you have an *onNavigate* event handler, call it from the *onDelete* event.

While the DBF (dBASE) table format supports soft deletes, in which the rows are only marked as deleted and not actually removed until the table is packed, there is no method in the data access classes to recall those records. Therefore a *delete*( ) should always be considered final.

The example attached to this topic shows how to use *delete*( ), in conjunction with the *setRange*( ) method, to delete all rows in a range or filter.

delete( ) [UpdateSet]

Topic group Related topics

Deletes the rows in the destination that are listed in the source.

Syntax

<oRef>.*delete*( )

<oRef>

The UpdateSet object that describes the delete.

Property of

UpdateSet

Description

delete( ) deletes the rows listed in the *source* rowset or table from the *destination* rowset or table. The *destination* must be indexed.

descending

Topic group

Determines the sort order, ascending or descending, of a specified index.

Property of

DBFIndex, Index

Description

An Index object's descending property determines whether the sort order for a key field is descending, the default setting, or ascending. In a "descending" order, character keys are ordered from Z to A, and then z to a; numeric keys are ordered from highest to lowest; and date keys are ordered from latest to earliest (a blank date is higher than all other dates).

destination

Topic group Related topics

The target rowset or table of an UpdateSet operation.

Property of

UpdateSet

Description

The *destination* property contains an object reference to a rowset or the name of a table that is the target of an UpdateSet operation. For an *append*( ), *update*( ), or *appendUpdate*( ), it refers to the rowset or table that is changed. For a *copy*( ), it refers to the rowset or table that receives the copies. If a table name is specified, that table is created, or overwritten if it already exists. For a *delete*( ), the *destination* property refers to the table from which rows are deleted.

The *source* property specifies the other end of the UpdateSet operation.

driverName

Topic group Related topics

The database driver used for the database connection.

Property of

Database

Description

The *driverName* property reflects the database driver used for the connection. It's determined by the database driver for the database's BDE alias and set automatically once the database is successfully made active.

For default databases, the *driverName* matches the System setting in the BDE Administrator.

dropIndex( )

Topic group Related topics

The *dropIndex*( ) method deletes an index for a specified table

Syntax

<oRef>.*dropIndex* (<tablename expC>,<indexName expC>)

<oRef>

The database in which the table exists.

<table name expC>

The name of the table containing the index

<indexname expC>

The index tag name

Property of

Database

```
ex. _app.databases[1].dropIndex("tablename","indexname")
```

dropTable( )

Topic group Related topics

Deletes (drops) a table from a database.

Syntax

<oRef>.*dropTable*(<table name expC>)

<oRef>

The database in which the table exists.

<table name expC>

The name of the table you want to delete.

Property of

Database

Description

dropTable( ) deletes a table and any existing secondary files, like memo files and indexes. *dropTable*( ) does not ask for confirmation; the deletion is immediate. The table cannot be open anywhere at the time of the *dropTable*( ); if it is, *dropTable*( ) fails.

To delete a Standard table, you can always use the default database in the default session by referring to it through the *databases* array property of the _app object. For example,

```
_app.databases[ 1 ].dropTable( "Temp" )
```

emptyTable( )

Topic group Related topics

Deletes all the rows in a table.

Syntax

<oRef>.*emptyTable*(<table name expC>)

<oRef>

The database in which the table exists.

<table name expC>

The name of the table you want to empty.

Property of

Database

Description

emptyTable( ) deletes all of the rows in a table, leaving an empty table structure, as if the table was just created. *emptyTable*( ) does not ask for confirmation; the deletion is immediate. The table cannot be open anywhere at the time of the *emptyTable*( ); if it is, *emptyTable*( ) fails.

To empty a Standard table, you can always use the default database in the default session by referring to it through the *databases* array property of the _app object. For example,

```
_app.databases[ 1 ].emptyTable( "YtdSales" )
```

endOfSet

Topic group Related topics

Specifies whether the row cursor is at the end-of-set.

Property of

Rowset

Description

The row cursor is always positioned at either a valid row or the end-of-set. There are two end-of-set positions: one before the first row and one after the last row. *endOfSet* is *true* if the row cursor is positioned at either end-of-set position.

When you first make a query active successfully, *endOfSet* is *true* if there are no rows that match the specified criteria in the query's SQL SELECT statement, or simply no rows in the tables selected.

When you apply a filter by calling *applyFilter*( ) or setting the *filter* property, *endOfSet* becomes *true* if there are no rows that match the filter criteria. Otherwise, the row cursor is positioned at the first matching row.

If you navigate backward before the first row in the set or after the last row in the set, this moves the row cursor to the end-of-set, so *endOfSet* becomes *true*. You can call the *first*( ) or *last*( ) methods to attempt to move the row cursor to the first or last row in the set. If after calling one of those methods, *endOfSet* is still *true*, then there are no visible rows in the current set.

Attempting to read a field value while at end-of-set returns a null value.

Attempting to change a field value while at end-of-set causes an error.

exactMatch

Topic group Related topics

Determines whether rowset searches are conducted using a partial string or an exact string match.

Property of

Rowset

Description

exactMatch allows you to determine what constitutes "equal to" when performing rowset searches. When *exactMatch* is set to "true", the default setting, field values in subsequent searches will be evaluated as a "match" only when they are identical to your search string.

When you set *exactMatch* to "false", a partial string, or "begins with" search is performed. Searching for the string "S", for example, will find "Smith" and evaluate it as a match.

execute( )

Topic group Related topics

Executes a query or stored procedure.

Syntax

This method is called implicitly by the Query or StoredProc object.

Property of

Query, StoredProc

Description

The *execute*( ) method executes a query or stored procedure. It is called implicitly after *prepare*( ) when you set the object's *active* property to *true*. In typical usage, you do not call this method directly.

Advanced applications may override the definition of this method to perform supplementary actions when executing the query or stored procedure. Custom data drivers must define this method to perform the appropriate actions to retrieve a rowset or call a stored procedure.

executeSQL( )

Topic group

Executes the specified SQL statement.

Syntax

<oRef>.*executeSQL*(<SQL expC>)

<oRef>

The database in which you want to execute the SQL statement.

<SQL expC>

The SQL statement.

Property of

Database

Description

Use *executeSQL*( ) to perform an SQL operation that does not have a data object equivalent, for example, to use data definition language (DDL) SQL where no rowset is desired, and for server-specific SQL.

fieldName

Topic group Related topics

The name of the field represented by the Field object.

Property of

Field (including DbfField, PdxField, SqlField)

Description

The *fieldName* property contains the name of the field that the Field object represents. The *fieldName* property is automatically filled in when the rowset object is generated.

For a calculated field, the *fieldName* contains the name of the field assigned when the Field object is created.

fields

Topic group Related topics

An array that contains the Field objects in a rowset.

Property of

Rowset, TableDef

Description

The *fields* property contains an object reference to the array of field objects in the rowset. These fields can be accessed by their field name or their ordinal position; for example, if *this* refers to a rowset:

```
this.fields[ "State" ].value = "CA" // Assign "CA" to State field
this.fields[ 1 ].value = 12 // Assign 12 to first field
```

To access the value of the field, you must reference the field's *value* property. You can use the *add*( ) method to add new Field objects to the *fields* array as calculated fields.

filename

Topic group Related topics

The name of the file that contains the desired data module.

Property of

DataModRef

Description

After setting the *filename* property to the file that contains the data module class definition, set the *dataModClass* property to the name of the desired class. Data modules are stored in files with a .DMD extension.

filter

Topic group Related topics

An SQL expression that filters out rows that do not match specified criteria.

Property of

Rowset

Description

A filter is a mechanism by which you can temporarily hide, or filter out, those rows that do not match certain criteria so that you can see only those rows that do match. The criteria is in the form of a character string that contains an SQL expression, like the one used in the WHERE clause of an SQL SELECT. Simple comparisons using the basic SQL comparison operators (=, <>, <, >, <=, >=) are supported; other predicates such as BETWEEN, IS NULL, IS NOT NULL and LIKE are not. Multiple comparisons may be joined by AND or OR. For example,

```
"Firstname = 'Waldo'"
```

In this case, you would see only those rows in the current rowset whose Firstname field was "Waldo". You can use the rowset's Filter mode, initiated by calling the *beginFilter*( ) method, to build the expression automatically, and then apply it with the *applyFilter*( ) method. The alternative is to assign the character string directly to the *filter* property.

If the filter expression contains a quoted string that contains an apostrophe, precede the apostrophe with a backslash. Note that the single quote used in SQL expressions for strings and the apostrophe are represented by the same single quote character on the keyboard. For example, if *this* is the rowset and you want to display rows with the Lastname "O'Dell":

```
this.filter := "Lastname = 'O\'Dell'"
```

Setting the *filter* property causes the row cursor to move to the first matching row. If no rows match the filter expression, the row cursor is moved to the end-of-set; the *endOfSet* property is set to *true*.

While a filter is active, the row cursor will always be at either a matching row or the end-of-set. Any time you attempt to navigate to a row, the row is evaluated to see if it matches the filter condition. If it does, then the row cursor is allowed to position itself at that row and the row can be seen. If the row does not match the filter condition, the row cursor continues in the direction it was moving to find the next matching row. It will continue to move in that direction until it finds a match or gets to the end-of-set. For example, suppose that *this* is the rowset, and you execute

the following to your program. If no filter is active, you would move four rows forward, toward the last row:

```
this.next( 4 )
```

If a filter is active, the row cursor will move forward until it has encountered four rows that match the filter condition, and stop at the fourth. That may be the next four rows in the rowset, if they all happen to match, or the next five, or the next 400, or never, if there aren't four rows after the current row that match. In that last case, the row cursor will be at the end-of-set.

In other words, when there is no filter active, every row is considered a match. By setting a filter, you filter out all the rows that don't match certain criteria.

To clear a filter, you can assign an empty string to the *filter* property, set the filter equal to *null*, or call the *clearFilter*( ) method.

In addition to using an SQL expression, you can filter out rows with more complex code by using the *canGetRow* event.

Note

When a field's *lookupSQL* property is set, and that field is referenced in the rowset's filter property, the value being compared by the filter is the field's true value, not the lookup value.

filterOptions

Topic group Related topics

Determines how values are matched for filtering.

Property of

Rowset

Description

The *filterOptions* property is an enumerated property that controls how the *value* properties in the field objects entered during Filter mode are matched against the values in the table. These are the options:

Value	Effect
0	Match length and case
1	Match partial length
2	Ignore case
3	Match partial length and ignore case

When matching partial length, the entire search value must match all or part of the value in the table, starting at the beginning of the field. For example, searching for "Central Park", will match "Central Park West", but "West" alone would not.

filterOptions also determines how fields are matched when specifying an SQL expression in the *filter* property.

The *filterOptions* property takes effect when you assign the SQL expression to the *filter* property or call *applyFilter*( ). Changing *filterOptions* after activating the filter has no effect (until you change the filter).

The default setting for *filterOptions* is "Match length and case".

findKey( )

Topic group Related topics

Finds the row with the exact matching key value.

Syntax

<oRef>.*findKey*(<exp> | <exp list>)

<oRef>

The rowset in which to do the search.

<exp>

The value to search for.

<exp list>

One or more expressions, separated by commas, to search for in a simple or composite key index for non-DBF tables.

Property of

Rowset

Description

findKey( ) performs an indexed search in the rowset, using the index specified by the rowset's *indexName* property. It looks for the first row in the index whose index key value matches <exp>, returning *true* or *false* to indicate whether a match is found.

findKey( ) is a navigation method; calling it fires the *canNavigate* event. If *canNavigate* returns *false*, *findKey*( ) does not attempt a search. If *canNavigate* returns *true*, and a search is attempted but fails, the row cursor remains at the current row and does not encounter an end-of-set. The *onNavigate* event always fires after a search attempt. For more information on how navigation methods interact with navigation events and implicit saves, see *next( )*.

findKey( ) always performs a partial key match with strings. For example, *findKey*("S") will find "Sanders", or whatever is the first key value that starts with the letter "S". To perform a full key match, pad <exp> with enough extra spaces to match the length of the index key value.

findKeyNearest( )

Topic group Related topics

Finds the row with the nearest matching key value.

Syntax

<oRef>.*findKeyNearest*(<exp> | <exp list>)

<oRef>

The rowset in which to do the search.

<exp>

The value to search for.

<exp list>

One or more expressions, separated by commas, to search for in a simple or composite key index for non-DBF tables.

Property of

Rowset

Description

findKeyNearest( ) performs an indexed search in the rowset, using the index specified by the rowset's *indexName* property. It looks for the first row in the index whose index key value matches <exp>, returning *true* or *false* to indicate if an exact match is found. If an exact match is not found, the row cursor is left at the nearest match; the row where the match would have been. For example, if "Smith" is followed by "Smythe" in the index, and the search expression is "Smothers", the search will fail and the row cursor will be left at "Smythe". "Smothers" comes after "Smith" and before "Smythe", so if it was in the index, it would be where "Smythe" is.

You can think of this exact, or nearest matching, as "equal, or the one after," as long as you remember that "after" depends on the index order. If the index is descending instead of ascending, then in the previous example, "Smythe" would be followed by "Smith", and a search for "Smothers" would end up on "Smith". The row cursor will end up on the end-of-set if the search value comes after the last value in the index.

findKeyNearest( ) is a navigation method; calling it fires the *canNavigate* event. If *canNavigate* returns *false*, *findKeyNearest*( ) does not attempt a search. *onNavigate* always fires after a search attempt. For more information on how navigation methods interact with navigation events and implicit saves, see *next( )*.

findKeyNearest( ) always performs a partial key match with strings. For example, *findKeyNearest*("Smi") will find "Smith". To perform a full key match, pad <exp> with enough extra spaces to match the length of the index key value.

first( )

Topic group Related topics

Moves the row cursor to the first row in the rowset.

Syntax

<oRef>.*first*( )

<oRef>

The rowset in which you want to move the row cursor.

Property of

Rowset

Description

Call *first*( ) to move the row cursor to the first row in the rowset. If a filter is active, it moves the row cursor to the first row in the rowset matching the filter criteria.

As a navigation method, *first*( ) interacts with *canNavigate*, *onNavigate*, and implicit saves. For more information, see *next( )*.

If a call to *first*( ) results in the *endOfSet* property returning *true*, either no rows remain that match the filter criteria or, if no filter is in use, no rows remain in the rowset.

Note

Using the rowset's *navigateByMaster* property to synchronize movement in master-detail rowsets, modifies the behavior of the *atLast*( ) method. See *navigateByMaster* for more information.

flush( )

Topic group Related topics Example

Commits data buffers to disk.

Syntax

<oRef>.*flush*( )

<oRef>

The rowset you want to write to disk.

Property of

Rowset

Description

When a row is saved, the changes are written to the rowset data buffer in memory. This buffer is written to disk only as needed; for example, before another block of rows are read into the buffer. This eliminates redundant disk writes that would slow your application.

flush( ) explicitly writes the rowset's data buffers to disk. Note that if a disk cache is active, the buffer is written to the disk cache; the cache decides when to actually write the data onto the physical disk.

refresh( ) is similar to *flush*( ) because in purging cached rows, *refresh*( ) writes any rows that have been changed but not yet committed to disk. *flush*( ) writes the rows, but does not purge the data buffer; the rows are still cached.

forExpression

Topic group

Limits the records included in the index to those meeting a specific condition.

Property of

DbfIndex

Description

Use the *forExpression* property to specify which rows in a set to include in the index by entering a condition that restricts the operation to certain rows. For example, entering, `COMPANY = "Santa Cruz Dry Goods",` restricts the index to rows for which the COMPANY field evaluates to the string "Santa Cruz Dry Goods".

getSchema( )

Topic group Related topics Example

Returns information about a database.

Syntax

<oRef>.*getSchema*(<item expC>)

<oRef>

The database you want to get information about.

<item expC>

The information to retrieve, which may be one of the following strings (which are not case-sensitive):

String	Information
DATABASES	A list of all databases aliases
PROCEDURES	A list of stored procedures defined in the database
TABLES	A list of all tables in the database
VIEWS	A list of all views in the database

Property of

Database

Description

Use *getSchema*( ) to get a list of all database aliases, or to get information about a specific database. Some databases may not support PROCEDURES or VIEWS. All lists are returned in an Array object; if the item is not supported, the array is empty.

Custom data drivers must define this method to return the appropriate information for their database.

goto( )

Topic group Related topics Example

Moves the row cursor to a specific row in the rowset.

Syntax

<oRef>.*goto*(<bookmark>)

<oRef>

The rowset in which you want to move the row cursor.

<bookmark>

The bookmark you want to move to.

Property of

Rowset

Description

Call *goto*( ) to move the row cursor to a specific row in the rowset. Store the current row position in a bookmark with the *bookmark*( ) method. Then you can return to that row later by calling *goto*( ) with that bookmark as long as the rowset has remained open. If the rowset has been closed, the bookmark is not guaranteed to return you to the correct row, since the table may have changed.

The bookmark uses the current index represented by the *indexName* property, if any. The same physical row in the table returns different bookmarks when different indexes are in effect. When you *goto*( ) a bookmark, the index that was in effect when the bookmark was returned is automatically activated.

If you attempt to *goto*( ) a row that is out-of-set, you will generate an error.

As a navigation method, *goto*( ) interacts with *canNavigate*, *onNavigate*, and implicit saves. For more information, see *next*( ).

handle [Data objects]

Topic group

The BDE handle of the object.

Property of

Database, Query, Rowset, Session, StoredProc

Description

The *handle* property represents the BDE handle for the object in question. The handle can be used if you want to call BDE functions directly.

indexes

Topic group Related topics Example

An array containing the table's index objects.

Property of

TableDef

Description

The TableDef's *indexes* property provides a means with which to view the properties associated with a table's array of index objects. As is true with all other TableDef properties, the *indexes* property is only intended to provide information about a tables indexes, and does not provide a means to further affect their values.

indexName [Data objects]

Topic group Related topics Example

The name of the index to use in the rowset.

Property of

Dbfindex, Index, Rowset

Description

indexName contains the name of the active controlling index tag for those table types that support index tags. It is set automatically when the query is activated to represent the tag used in the SQL SELECT's ORDER BY clause, if the ORDER BY is satisfied by an index. Assigning a new value to *indexName* supersedes any ORDER BY designated in the SQL SELECT statement.

For tables with primary keys, a blank *indexName* indicates that the primary key is the controlling index.

The index tag is also used in a master-detail link. The index tag of the detail rowset must match the field or fields specified in the *masterFields* property.

When specifying an *indexName* for data in a report, be sure to set the report's *autoSort* property to *false* to prevent the report from modifying the SQL statement. The modified SQL statement may generate a temporary result set that has no indexes; attempting to designate an *indexName* would cause an error.

indexName [UpdateSet]

Topic group Related topics

The name of the index to use for indexed UpdateSet operations.

Property of

UpdateSet

Description

The *destination* rowset or table must be indexed for the *update*( ), *appendUpdate*( ), and *delete*( ) operations. The *indexName* property specifies the key or tag name that is to be used. For tables with primary keys, the primary key is used by default. Set the *indexName* property only if you want to use another key. For DBF (dBASE) tables, you must specify an index tag name.

isolationLevel

Topic group Related topics

Determines the isolation level of a transaction.

Property of

Database

Description

The *isolationLevel* property is an enumerated property that determines the isolation level of a transaction. It applies to SQL-server database transactions only. For Standard table transactions, it has no effect. These are the options:

Value	Effect
0	Read uncommitted
1	Read committed
2	Repeatable read

The default is Read committed.

isRowLocked( )

Topic group Related topics

Returns a logical value indicating whether the current rowset has locked the current row.

Syntax

<oRef>.*isRowLocked*( )

<oRef>

An object reference to the rowset.

Property of

Rowset

Description

Use *isRowLocked*( ) to determine if the same instance of the current row, in the current rowset, is locked before an attempt is made to edit or delete a record. *isRowLocked* returns true to indicate the row is locked and false to indicate it's not.

The *isRowLocked*( ) method only returns information about the current instance of a rowset. When dealing with multiple instances of a row or rowset, you'll need to attempt an explicit row lock with the *lockRow*( ) method.

isSetLocked( )

Topic group Related topics

Returns a logical value indicating whether the current rowset is locked.

Syntax

<oRef>.*isSetLocked*( )

<oRef>

An object reference to the rowset.

Property of

Rowset

Description

Use *isSetLocked*( ) to determine if the same instance of the current rowset is locked before an attempt is made to edit or delete. *isSetLocked*( ) returns true to indicate the rowset is locked and false to indicate it isn't.

The *isSetLocked*( ) method only returns information about the current instance of a rowset. When dealing with multiple instances of a rowset, you'll need to attempt an explicit rowset lock with the *lockSet( )* method.

keyViolationTableName

Topic group Related topics

Name of the table in which you want to collect rows that could not be added because they would have caused a key violation.

Property of

UpdateSet

Description

In tables with primary keys, only one row in the table may have a particular primary key value. If the row to be added during an *append*( ) contains a key value that is the same as an already-existing primary key, that row cannot be added to the table, since it would have caused a primary key violation. Instead of being added to the *destination* rowset or table, that row is copied to the table specified by the *keyViolationTableName* property.

language

Topic group Related topics

The Language Driver currently being used to access a table

Property of

TableDef

Description

Returns a character string indicating the name of the current Language Driver. The value returned will reflect the language version selected during installation, or specified through the BDE. Read-only.

languageDriver

Topic group Related topics

The Language Driver currently being used to access a table

Property of

Rowset

Description

Returns a character string indicating the name of the current Language Driver. Read-only.

last( )

Topic group Related topics

Moves the row cursor to the last row in the rowset.

Syntax

<oRef>.*last*( )

<oRef>

The rowset in which you want to move the row cursor.

Property of

Rowset

Description

Call *last*( ) to move the row cursor to the last row in the rowset. If a filter is active, it moves the row cursor to the last row in the rowset that matches the filter criteria.

As a navigation method, *last*( ) interacts with *canNavigate*, *onNavigate*, and implicit saves. For more information, see *next( )*.

If a call to *last*( ) results in the *endOfSet* property returning *true*, either no rows remain that match the filter criteria or, if no filter is in use, no rows remain in the rowset.

Going to the last row in a rowset may not be an optimized operation on some SQL servers. For those servers, calling *last*( ) may take a long time for large rowsets.

Note

Using the rowset's *navigateByMaster* property to synchronize movement in master-detail rowsets, modifies the behavior of the *atLast*( ) method. See *navigateByMaster* for more information.

length

Topic group Related topics

The maximum length of the field.

Property of

Field

Description

A field's length represents the number of bytes used in the table for that field, and for character and numeric fields, the maximum length of the item that it can store.

For character fields, the *length* property represents the maximum number of characters in the string. Attempting to store more characters in that field results in the string being truncated.

For numeric fields, the *length* property represents the maximum number of characters in the number, including the digits, and any sign or decimal point. Attempting to store a number with more digits than the maximum results in numeric overflow, in which the actual value of the number is lost, and is simply considered to be bigger than the maximum allowed; it is usually represented by a string of asterisks.

live

Topic group Related topics

Specifies whether the rowset can be modified.

Property of

Rowset

Description

Before making a query active, you can determine whether the rowset that is generated is editable or not. You can choose to make it not editable to prevent accidental modification of the data.

locateNext( )

Topic group Related topics

Applies the locate criteria again to search for another row.

Syntax

<oRef>.*locateNext*([<rows expN>])

<oRef>

The rowset in which to move the row cursor.

<rows expN>

The Nth row to find. By default, the next row forward.

Property of

Rowset

Description

When the *applyLocate*( ) method is called, it moves the row cursor to the first row that matches the locate criteria. From then on, you can move forward and backward to other rows that match the same criteria by calling *locateNext*( ).

locateNext( ) takes an optional numeric parameter that specifies in which direction, forward or backward, to look and at which match to stop, relative to the current row position. A negative number indicates a search backward, toward the first row; a positive number indicates a search forward, toward the last row. For example, a parameter of –3 means to look backward from the current row to find the third matching row.

If the row cursor encounters the end-of-set before the desired match is found, the search stops, leaving the row cursor at the end-of-set.

As a navigation method, *locateNext*( ) interacts with *canNavigate*, *onNavigate*, and implicit saves. For more information, see *next*( ).

locateNext( ) returns *true* to indicate that the desired match was found and *false* to indicate that it wasn't.

locateOptions

Topic group Related topics

Determines how values are matched for locating.

Property of

Rowset

Description

The *locateOptions* property is an enumerated property that controls how the *value* properties in the field objects entered during Locate mode are matched against the values in the table. These are the options:

Value	Effect
0	Match length and case
1	Match partial length
2	Ignore case
3	Match partial length and ignore case

When matching partial length, the entire search value must match all or part of the value in the table, starting at the beginning of the field. For example, searching for "Century City", will match "Century City East", but "East" alone would not.

locateOptions also determines how fields are matched when using an SQL expression with the *applyLocate*( ) method.

The default setting for *locateOptions* is "Match length and case".

lock

Topic group Related topics Example

The date and time of the last successful lock made to the row.

Property of

LockField

Description

Use *lock* after a failed lock attempt to determine the date and time of the current lock that is blocking your lock attempt. The date and time are represented in a string in the following format:

```
MM/DD/YY HH:MM:SS
```

This format is accepted by the constructor for a Date object, so you can easily convert the *update* string into an actual date/time.

This property is available only for DBF tables that have been CONVERTed.

lockRetryCount

Topic group Related topics

The number of times to retry a lock attempt.

Property of

Session

Description

Any attempt to change the data in a row, for example, typing a letter in a *dataLink*ed Entryfield control, causes an automatic row lock to be attempted. In addition to the automatic row locking, you may request an explicit row or rowset lock with the *lockRow*( ) and *lockSet*( ) methods.

If someone else already has a conflicting lock, the initial lock attempt fails. The *lockRetryCount* property indicates the number of times the lock attempt will be retried, while the *lockRetryInterval* indicates the number of seconds to wait between each attempt. If after all the attempts the lock has not been secured, the lock request fails.

lockRetryInterval

Topic group Related topics

The number of seconds to wait between each lock retry attempt.

Property of

Session

Description

Any attempt to change the data in a row, for example, typing a letter in a *dataLink*ed Entryfield control, causes an automatic row lock to be attempted. In addition to the automatic row locking, you may request an explicit row or rowset lock with the *lockRow*( ) and *lockSet*( ) methods.

If someone else already has a conflicting lock, the initial lock attempt fails. The *lockRetryCount* property indicates the number of times the lock attempt will be retried, while the *lockRetryInterval* indicates the number of seconds to wait between each attempt. If after all the attempts, the lock has not been secured, the lock request fails.

lockRow( )

Topic group Related topics Example

Attempts to lock the current row.

Syntax

<oRef>.*lockRow*( )

<oRef>

The rowset in which you want to lock the current row.

Property of

Rowset

Description

An automatic row lock is attempted whenever the *value* property of a Field object is modified, either directly by assignment, or indirectly through a *dataLink*ed control.

You may use *lockRow*( ) to attempt an explicit row lock. Whether the lock is automatic or explicit, it will fail if the current row or the entire rowset is already locked.

lockRow( ) returns *true* to indicate that the lock was successful and *false* to indicate that it wasn't.

Row locking support varies among different table types. The Standard (DBF and DB) tables fully support row locking; most SQL servers do not. For servers that do not support true locks, the Borland Database Engine emulates optimistic locking. Any lock request is assumed to succeed. Later, when the actual attempt to change the data occurs, if the data has changed since the lock attempt, an error occurs.

lockSet( )

Topic group Related topics

Attempts to lock the entire rowset.

Syntax

<oRef>.*lockSet*( )

<oRef>

The rowset you want to lock.

Property of

Rowset

Description

You may use *lockSet*( ) to attempt to lock the entire rowset. The rowset cannot be locked if someone else already has any other row or set locks on the rowset.

Set locks are session-based. Once a *lockSet*( ) attempt succeeds, all other *lockSet*( ) requests for the same set from rowsets in queries assigned to the same session will succeed. Query objects must be assigned to different Session objects for set locking to work properly.

Locking the rowset is not the same as accessing the table exclusively. Exclusive access means that you are the only one who has the table open. In contrast, locking a rowset allows others to view, but not modify, the rowset.

lockSet( ) returns *true* to indicate that the lock was successful and *false* to indicate that it wasn't.

Set locking support varies among different table types. The Standard (DBF and DB) tables fully support set locking, as do a few SQL servers. For servers that do not support true locks, the Borland Database Engine emulates optimistic locking. Any lock request is assumed to succeed. Later, when the actual attempt to change the data occurs, if the data has changed since the lock attempt, an error occurs.

lockType

Topic group Related topics

Determines whether or not explicit locks can be released by a call to rowset.save().

Property of

Rowset

Description

Allowed values for lockType are:

0 - Automatic = row locks obtained by calling rowset.lockrow() are released by calls to Save() or Abandon()

1 - Explicit = row locks obtained by calling rowset.lockrow() are NOT released by calls to Save() or Abandon()

The default for lockType is 0 - Automatic unless an overriding setting is set In plus.ini or the application's .ini file.

ini file setting example:

```
[Rowset]
LockType=0     (or)
LockType=1
```

Allows user to set default rowset.lockType via ini setting.

logicalSubType

Topic group Related topics

A database independent name indicating the data subtype of the value stored in a field.

Property of

CalcField, DbfField, Field, PdxField, SqlField

Description

Each database engine has its own set of data types that are referred to as its native data types. A data type in one database engine may be physically identical to a data type used by another database engine, but have a different name.

Mapping these native types to a set of database independent (logical) data types allows physically identical data types to have the same *logicalType* even when their native data types differ.

For example, the logical type for both a Paradox table's "Alpha" field and a dBASE table's "Character" field is "ZSTRING". This indicates they are both character strings with a null byte at the end of the string.

Some *logicalTypes* contain sub groupings, called *logicalSubTypes*, which specify further the type of data that can be stored in each field type. It may be necessary, therefore, to also compare *logicalSubTypes* when checking for data type compatibility.

For example, a BLOB *logicalType* may also contain one of the following *logicalSubTypes*:

MEMO
BINARY
FMTMEMO
OLEOBJ
GRAPHIC
DBSOLEOBJ
TYPEDBINARY
ACCOLEOBJ

The following table lists possible values for the *logicalSubType* property:

logicalType	logicalSubType	Description
FLOAT	MONEY	Money
BLOB	MEMO	Text memo
	BINARY	Binary data
	FMTMEMO	Formatted text
	OLEOBJ	OLE object (Paradox)
	GRAPHIC	Graphics object
	DBSOLEOBJ	dBASE OLE object
	TYPEDBINARY	Typed binary data
	ACCOLEOBJ	Access OLE object
ZSTRING	PASSWORD	Password
	FIXED	CHAR type
	UNICODE	Unicode
INT32	AUTOINC	Auto Increment value

Tip: Using the *logicalType* and *logicalSubType* properties, you could write a dBASE Plus program to check whether data from a table containing a DbfField data type can be copied to a table containing a PdxField data type.

logicalType

Topic group Related topics

A database independent name indicating the data type of the value stored in a field.

Property of

CalcField, DbfField, Field, PdxField, SqlField

Description

Each database engine has its own set of data types that are referred to as its native data types. A data type in one database engine may be physically identical to a data type used by another database engine, but have a different name.

Mapping these native types to a set of database independent (logical) data types allows physically identical data types to have the same *logicalType* even when their native data types differ.

For example, the logical type for both a Paradox table's, "Alpha", field and a dBASE table's, "Character", field is "ZSTRING". This indicates they are both character strings with a null byte at the end of the string.

Note: The Field object's type property contains the native type of a field.

The following table lists possible values for the *logicalType* property:

logicalType	Description
UNKNOWN	
ZSTRING	Null terminated character string
DATE	Date (32 bit)
BLOB	Short for, "**b**inary **l**arge **ob**ject", a collection of binary data stored as a single entity in a database management system.
BOOL	Boolean
INT16	16 bit signed integer
INT32	32 bit signed integer
FLOAT	64 bit floating point
BCD	Binary Coded Decimal
BYTES	Fixed number of bytes
TIME	Time (32 bit)
TIMESTAMP	Time-stamp (64 bit)
UINT16	Unsigned 16 bit integer
UINT32	Unsigned 32 bit integer
FLOATIEEE	80 bit IEEE float
VARBYTES	Length prefixed string of bytes
LOCKINFO	Lock for LOCKINFO typedef
CURSOR	For Oracle Cursor type

Tip: Using the *logicalType* property, you could write a dBASE Plus program to check whether data from a table containing a DbfField data type can be copied to a table containing a PdxField data type.

login( )

Topic group Related topics Example

Logs in user to DBF table security for a session.

Syntax

<oRef>.*login*(<group name expC>, <user name expC>, <password expC>)

<oRef>

The session to log into.

<group name expC>

The group name.

<user name expC>

The user name.

<password expC>

The password.

Property of

Session

Description

DBF table security is session-based. All queries assigned to the same session in their *session* property have the same access level.

If someone attempts to open an encrypted table and has not logged in to the session, they will be prompted for the group name, user name, and password. Responding attempts to log the user into the session.

The *login*( ) method allows you to log in to the session directly. You can do this if you're assigning a default access level, so that users won't be prompted; or if you're writing your own custom login form, in which case you will need to call *login*( ) with the returned values.

login( ) returns *true* or *false* to indicate whether the login was successful.

loginDBAlias

Topic group Related topics

The currently active database alias, or BDE alias, from which to obtain login credentials (user id and password) to be used in activating an additional connection to a database.

Property of

Database

Description

The *loginDBAlias* property can be used to setup additional connections to a database without having to prompt the user each time for login credentials.

The default value for the *loginDBAlias* property is an empty string.

Using the *loginDBAlias* property

1. Create a database object.
2. Set the *databaseName* property of the new database object to the appropriate database alias.
3. From an already active database object, assign the value from its *databaseName* property to the new database object's *loginDBAlias* property.
4. Set *active* to true on the new database object.

When activating the new database object, dBASE will lookup the user id and password used to login to the already active database object and submit them to the database engine in the same way it submits a *loginString*.

If the user id and password are valid, the user will not be prompted to enter any login credentials for the new database object.

loginString

Topic group Related topics

The user name and password to use to log in to a database.

Property of

Database

Description

Some databases require that you log in to them to access their tables. When you set the Database object's *active* property to *true* to open the connection, a login dialog will appear, prompting the user for the user name and password.

You can prevent the login dialog from appearing by setting the *loginString* property to a string containing a valid user name and password of the form "userName/password". If the user name and password provided through *loginString* are not valid, the login dialog will appear when you attempt to activate the database.

lookupRowset

Topic group Related topics Example

The rowset containing lookup values for a field.

Property of

Field

Description

Use *lookupSQL* or *lookupRowset* to implement automatic lookups for a field. For information on how automatic lookups work, see *lookupSQL*.

The simpler implementation is to set the *lookupSQL* property. This automatically generates a lookup rowset, which you can reference through the *lookupRowset* property.

The more advanced technique is to generate your own lookup rowset, which must follow the same structure as detailed for *lookupSQL*. Then assign a reference to this rowset to the *lookupRowset* property. Doing so releases any internal rowset generated for *lookupSQL*, if any. This technique might be used if you want to use the same lookup for multiple fields.

lookupSQL

Topic group Related topics Example

An SQL SELECT statement describing a rowset that contains lookup values for a field.

Property of

Field

Description

Use *lookupSQL* or *lookupRowset* to implement automatic lookups for a field. When a control that supports lookups, like the ComboBox control, is *dataLink*ed to a field with either *lookupSQL* or *lookupRowset* defined, the control will:

Populate itself with display values from the lookup rowset
Lookup the true value of the field in the lookup rowset
Display the corresponding lookup value in the control
Do the reverse lookup when the display value in the control is changed
Write the corresponding true value back to the field

If the display lookup fails, a blank is displayed in the control. If the reverse lookup fails, a *null* is written to the field.

The same automatic lookups are applied when accessing the *value* property of the field. The *value* of the field will appear to be the lookup value. Assigning to the *value* will perform the reverse lookup.

Setting the *lookupSQL* property is the simpler way of implementing automatic lookups. *lookupSQL* contains an SQL statement of the form:

```
SELECT <lookup field>, <display field> [,...] FROM <lookup table> [<options>]
```

The first two fields must be the lookup field and the display field, respectively. The display field may be a calculated field. You may include other fields so that you can get information about the chosen row. The SQL SELECT statement may include the usual options; in particular, you may want the table to be ordered on the lookup field (or use a table where such an index is available) for faster lookups. The SQL statement is executed in the same database as the query (or stored procedure) that contains field's rowset.

When an SQL statement is assigned to *lookupSQL*, the *lookupRowset* property will contain a reference to the generated rowset. You may refer to the fields in the matched lookup row through this reference. For advanced applications, you may assign your own rowset to *lookupRowset*. This releases the generated rowset.

Note

When a field's *lookupSQL* property is set, and that field is referenced in the rowset's filter property, the value being compared by the filter is the field's true value, not the lookup value.

lookupTable

Topic group Related topics

The table used for a DB (Paradox) field's lookup.

Property of

PdxField

Description

lookupTable contains the name of the lookup table used to assist in the filling in of the field represented by the PdxField object. For more information on Paradox table lookups, see *lookupType*.

lookupType

Topic group Related topics

The type of lookup used by a DB (Paradox) field.

Property of

PdxField

Description

lookupType specifies the type of lookup used to assist in the filling in of the field represented by the PdxField object. It is an enumated property that can have one of the following values:

Value	Description
0	No lookup
1	Lookup field only, no help
2	Lookup and fill all corresponding fields, no help
3	Lookup field only, with help
4	Lookup and fill all corresponding fields, with help

dBASE Plus does not support the user interface required for Paradox lookup help. Also, validity checking is not performed whenever all corresponding fields are filled; this is so that (in Paradox) you can substitute the field value with the value of a same-named field in the lookup table that is not the lookup field.

Therefore, the only support for Paradox lookups in *dBASE Plus* is for validity checking; to make sure the value stored in the field is listed in the lookup field in the lookup table, and only when *lookupType* is set to 1 or 3. For example, a Customer ID field in an Orders table can check that the Customer ID is listed in the Customer table. An attempt to store an unlisted value in the field results in a database engine-level exception.

Consider using the automatic lookup provided by *lookupSQL* and *lookupRowset* instead.

masterChild

Topic group Related topics

Specifies whether the rows in a child table are constrained to only those rows matching the key value from a row in the parent table.

Property of

Rowset

Description

The *masterChild* property is set in the detail rowset (child table in a master-child relation).

masterChild can be set to either:

0 – Constrained (default)
1 – UnConstrained

When constrained, the child table in a relation is filtered so that only rows that match the key value from a row in a parent table can be navigated to and displayed in a data object. If no child rows match the current parent row, then no child rows can be navigated to or shown in a data object.

When unconstrained, navigating in a parent table triggers any child tables to be positioned at the first child row that matches the parent row. All child rows can still be navigated to and displayed in a data object. If no matching child rows exist for the current parent row, the child table is positioned to the last record for the current index order.

The *masterChild* property is ignored if the *masterRowset* and *masterFields* rowset properties have not been set and a link established to the parent table.

masterFields

Topic group Related topics Example

A list of fields in the master rowset that link it to the detail rowset.

Property of

Rowset

Description

The *masterFields* property is set in the detail rowset. It is a string that contains a list of fields in the master rowset that are matched against the detail rowset's active controlling index, as

specified by the *indexName* property. By setting the property in the detail rowset, one master rowset can control multiple detail rowsets.

The *masterRowset* property should be set before *masterFields*. Once *masterFields* is set, by default, the detail rowset is constrained to show the detail rows that match the current row in the master rowset. To override the constraint, set the rowset's masterChild property to 1 – Unconstrained.

You may cancel the master-detail link by setting either property to an empty string.

For table formats that support multi-field indexes (DBF does not—it uses expression indexes instead), multiple fields in the *masterFields* list are separated by semicolons.

You may link the rowsets through an expression by creating a calculated field in the master rowset and using that calculated field name in the *masterFields* list.

masterRowset

Topic group Related topics Example

A reference to the master rowset that is linked the detail rowset.

Property of

Rowset

Description

The *masterRowset* property is set in the detail rowset. It is an object reference to the master rowset that constrains the detail rowset. By setting the property in the detail rowset, one master rowset can control multiple detail rowsets.

The *masterRowset* property should be set before *masterFields*. Once *masterFields* is set, the detail rowset is constrained to show the detail rows that match the current row in the master rowset. To override the constraint, set the detail rowset's masterChild property to 1 – Unconstrained.

You may cancel the master-detail link by setting either property to an empty string.

masterSource

Topic group Related topics Example

A reference to the rowset that acts as the master in a master-detail link and provides parameter values.

Property of

Query

Description

Use *masterSource* to create a master-detail link between two queries where parameters are used in the detail query. *masterSource* is assigned a reference to the *rowset* in the master query.

By setting the *masterSource* property, the parameters in the SQL statement are automatically substituted with matching fields from the master rowset, thereby constraining the detail query. Calculated fields may be used. The fields are matched to the parameters by name. The field name match is not case-sensitive.

As navigation occurs in the *masterSource* rowset, the parameter values are resubstituted and the detail query is requeried.

An alternate approach to creating a master-detail link is through the *masterRowset* and *masterFields* properties. While *masterRowset* and *masterFields* are used to link one rowset to another using an index and matching field values, *masterSource* creates a query-to-rowset link between the parameters in the detail query and the master rowset.

maximum

Topic group Related topics

The maximum allowed value of a field.

Property of

DbfField, PdxField

Description

maximum specifies the maximum allowed value of the field represented by the field object. A blank value indicates no maximum. The *maximum* is the same data type as the field, except for numeric fields that have no *maximum*; in that case, *maximum* is *null*.

Only character, date, and numeric fields (all variations) have a *maximum*. DBF tables must be level 7 to support *maximum*.

If you *dataLink* a SpinBox component to a field with a *maximum*, that value becomes the default *rangeMax* property of that component.

minimum

Topic group Related topics

The minimum allowed value of a field.

Property of

DbfField, PdxField

Description

minimum specifies the minimum allowed value of the field represented by the field object. A blank value indicates no minimum. The *minimum* is the same data type as the field, except for numeric fields that have no *minimum*; in that case, *minimum* is *null*.

Only character, date, and numeric fields (all variations) have a *minimum*. DBF tables must be level 7 to support *minimum*.

If you *dataLink* a SpinBox component to a field with a *minimum*, that value becomes the default *rangeMin* property of that component.

modified

Topic group Related topics Example

A flag to indicate whether the current row has been modified.

Property of

Rowset

Description

The *modified* property indicates whether the current row has been modified. It is automatically set to *true* whenever the *value* of any Field object is changed, either directly by assignment, or indirectly through a *dataLink*ed control.

If *modified* is *true*, then an attempt to save the row is made if there is navigation off the row or a *state* switch in the rowset. If *modified* is *false*, then this implicit save is not attempted.

modified is set to *false* whenever a row is read into the row buffer after navigating to it, is refreshed by *refreshRow*( ) or *refresh*( ), or is saved. You may also set the *modified* property to *true* or *false* manually. For example, you can set *modified* to *false* after assigning some *value* properties during an *onAppend* event. This makes the values you filled in default values, and the row will not be automatically saved if the user does not add more information.

In addition to tracking changes during normal data entry, the *modified* property is also set to *true* during Filter and Locate modes. This allows you to determine if any criteria have been specified before attempting an *applyFilter*( ) or *applyLocate*( ). When in either of these modes, navigation cancels the mode and moves the row cursor relative to the last row position, but no save is attempted, even if *modified* is *true*.

name [Data object]

Topic group

claThe name of a custom data object

Property of

All Data object classes

Description

The Data object *name* property simply identifies the name associated with a particular Data Object. This property is read-only and is assigned when the object is created.

navigateByMaster

Topic group Related topics

Use the *navigateByMaster* property to flag a detail rowset to move when its master rowset is moved. The *navigateByMaster* property allows detail rowsets and a linked master rowset to be navigated as though they were part of a single, combined rowset (similar to the xDML SET SKIP command).

Property of

Rowset

Description

When set to true in a detail rowset, *navigateByMaster* signals the linked master rowset to navigate through any matching detail rows before moving to a new master row.

More specifically, *navigateByMaster* :

Flags a detail rowset so its row cursor is moved when its master rowset's *next*( ), *first*( ), or *last*( ) methods are called

Flags a detail rowset so its *atFirst*( ) or *atLast*( ) methods are called when its master rowset's *atFirst*( ) or *atLast*( ) methods are called.

When a master rowset has one or more detail rowset's with *navigateByMaster* set to true, the behavior of the following rowset methods is modified as follows:

first( ) Ensures that linked detail rowsets are positioned to the first row matching the first master row. After positioning a master rowset to its first row, the masters *first*( ) method positions any linked detail rowsets to their first matching row, which in turn position any linked grandchild rowsets to their first row matching their master rowsets. This process continues recursively through the entire tree of linked rowsets.

next( ) Attempts to move detail and master rowsets such that they appear to have moved one or more rows relative to their starting positions, as if they were a single combined rowset. *next*( ) can be called with an optional numeric parameter specifying the direction (positive to move forward, negative to move backward) and number of rows to move. If no parameter is specified, *next*( ) defaults to moving one row forward. *next*( ) will return *true* if it is able to move the number of rows specified, otherwise it returns *false*.

next( ) moves the row cursors according to the following rules:

next( ) will only move linked detail rowsets that have their *navigateByMaster* property set to *true*.

next( ) will attempt to move these linked detail rowsets before moving the master rowset.

If a rowset has more than one linked detail rowset, *next*( ) will attempt to move them in the order in which they were linked to the master rowset. In addition, only detail rows matching the current master row will be moved (i.e. navigation occurs as if the master-detail link is constrained)

When moving in a detail rowset, *next*( ) will continue moving in the same detail rowset until it moves the number of rows requested, or it reaches end-of-set (i.e. no more detail rows are found matching the current master row). If the detail rowset has reached end-of-set, and there are still more rows to be moved, *next*( ) will continue with the next linked detail rowset or, if there are no other linked detail rowsets, *next*( ) will move the master rowset one row and synchronize the linked detail rowsets to:

their first matching row (if moving forward)

their last matching row (if moving backward)

end-of-set (if no matching row is found).

If there are still more rows to be moved to, *next*( ) will repeat this process starting once again with the first linked detail rowset.

If a linked detail rowset , for example d1, is itself a master rowset and has its own detail rowset , d2, (with *navigateByMaster* set to true), it will act as a master rowset and follow the same sequence of events described above. The net result of this sequence is that the lowest detail rowset (d2 in this example) will be moved first. When d2 reaches end-of-set, its master rowset, d1, will be moved. When d1 reaches end-of-set, its master rowset will be moved.

last( ) Ensures that linked detail rowsets are positioned to the last row matching the last master row. After positioning a master rowset to its last row, the master's *last*( ) method positions any linked detail rowsets to their last matching row, which in turn position any linked grandchild rowsets to their last row matching their master rowsets. This process continues recursively through the entire tree of linked rowsets.

atFirst( ) Returns *true* when a master rowset is at the first row and all linked detail rowsets, whose *navigateByMaster* properties are set to *true*, are at their first matching rows. Otherwise returns *false*.

atLast( ) Returns *true* when a master rowset is at the last row and all linked detail rowsets, whose *navigateByMaster* properties are set to *true*, are at their last matching rows. Otherwise returns *false*.

The *navigateByMaster* property's default is *false*.

To use this property:

In the detail rowset, set *navigateByMaster* to *true*

Specify the master rowset for the form.rowset (using the standard toolbar's navigation buttons).

Specify the master rowset as the grid's datalink if you want to setup a grid containing columns from both the master and linked detail rowsets.

Set the grandchild rowset's *navigateByMaster* to *true* to add additional master detail levels (such as parent, child, grandchild):

Grid and Browse classes

dBASE Plus's Grid class now provides correct rowset navigation when datalinked to a master rowset with at least one detail rowset whose *navigateByMaster* is set to *true*.

Similarly, *dBASE Plus*'s Browse class provides correct navigation when controlled by a table using xDML SET RELATION and SET SKIP commands.

navigateMaster

Topic group Related topics

Allows a linked-detail rowset to affect movement in its master rowset so that master and detail rowsets are navigated as though they were part of a single, combined rowset (similar to the xDML SET SKIP command).

Property of

Rowset

Description

When a detail rowset's *next*( ) method reaches end-of-set, after having been called explicitly with it's *navigateMaster* property set to *true*, it will move its master rowset to the next row in the master rowset's current order.

The *navigateMaster* property's default is *false*.

next( )

Topic group Related topics

Moves the row cursor to another row relative to the current position.

Syntax

<oRef>.*next*([<rows expN>])

<oRef>

The rowset in which you want to move the row cursor.

<rows expN>

The number of rows you want to move. By default, the next row forward.

Property of

Rowset

Description

next( ) takes an optional numeric parameter that specifies in which direction, forward or backward, to move and how many rows to move through, relative to the current row position. A negative number indicates a search backward, toward the first row; a positive number indicates a search forward, toward the last row. For example, a parameter of 2 means to move forward two rows.

If a filter is active, it is honored.

If the row cursor encounters the end-of-set while moving, the movement stops, leaving the row cursor at the end-of-set, and *next*( ) returns *false*. Otherwise *next*( ) returns *true*.

Navigation methods such as *next*( ) will cause the rowset to attempt an implicit save if the rowset's *modified* property is *true*. The order of events when calling next( ) is as follows:

1. If the rowset has a *canNavigate* event handler, it is called. If not, it's as if *canNavigate* returns *true*.
    - If the *canNavigate* event handler returns *false*, nothing else happens and *next*( ) returns *false*.
    - If the *canNavigate* event handler returns *true*, the rowset's *modified* property is checked.
    - If *modified* is *true*:
        - The rowset's*canSave* event is fired. If there is no *canSave* event, it's as if *canSave* returns *true*.
        - If *canSave* returns *false*, nothing else happens and *next*( ) returns *false*.
        - If *canSave* returns *true*, *dBASE Plus* tries to save the row. If the row is not saved, perhaps because it fails some database engine-level validation, a DbException occurs—*next*( ) does not return.
        - If the row is saved, the *modified* property is set to *false*, and the *onSave* event is fired.
    - After the current row is saved (if necessary):
        - The row cursor moves to the designated row.
        - The *onNavigate* event fires.
        - *next*( ) returns *true* (if the navigation did not end up at the end-of-set).

Other navigation methods go through a similar chain of events.

Note

Using the rowset's *navigateByMaster* property to synchronize movement in master-detail rowsets, modifies the behavior of the *next*( ) method. See *navigateByMaster* for more information.

notifyControls

Topic group Related topics

Specifies whether *dataLink*ed controls are updated as field values change or the row cursor moves.

Property of

Rowset

Description

notifyControls is usually *true* so that *dataLink*ed controls are automatically updated as you navigate from row to row or when you directly assign values to the *value* property of Field objects.

You may set *notifyControls* to *false* if you are performing some data manipulation and don't want the overhead of constantly updating the controls.

When *notifyControls* is set to *true*, the controls are always refreshed, as if *refreshControls*( ) was called.

onAbandon

Topic group Related topics Example

Event fired after the rowset is successfully abandoned.

Parameters

none

Property of

Rowset

Description

A rowset may be abandoned explicitly by calling its *abandon*( ) method, or implicitly via the user interface by pressing Esc or choosing Abandon Row from the default Table menu or toolbar while editing table rows. While the *canAbandon* event fires first to see if the abandon actually takes place, *onAbandon* fires after the abandon occurs.

If you are abandoning changes made to a row, the row is automatically refreshed, so there is no need to call *refreshRow*( ) in the *onAbandon*. However, this is not considered navigation, so if you have an *onNavigate* event handler, you should call it from *onAbandon*.

onAppend

Topic group Related topics Example

Event fired after the rowset successfully enters Append mode.

Parameters

none

Property of

Rowset

Description

A rowset may be put in Append mode explicitly by calling its *beginAppend*( ) method, or implicitly via the user interface by choosing Append Row from the default Table menu or toolbar while editing table rows. While the *canAppend* event fires first to see if the new append actually takes place, *onAppend* fires after the row buffer has been cleared and is ready for new values.

You can use *onAppend* to do things like automatically time stamp the new row or fill in default values. If you use *onAppend* to set field values, set the *modified* property to *false* at the end of the event handler to indicate that the row hasn't been changed by the user. This way, if the user does not add any more data, the row will not be saved automatically if they navigate to another row or try to append another.

onChange

Topic group Related topics Example

Event fired after a field's *value* property is successfully changed.

Parameters

none

Property of

Field (including DbfField, PdxField, SqlField)

Description

A Field object's *value* property may be changed directly by assigning a value to it, or indirectly through a *dataLink*ed control. When assigning a value, the change occurs during the assignment statement. When using a *dataLink*ed control, the change doesn't happen until the user tries to move the focus to another control. In both cases, *canChange* fires first to see if the change can actually take place. If it does, the value is changed and then *onChange* is fired.

onClose

Topic group Related topics

Event fired after a query or stored procedure is successfully closed.

Parameters

none

Property of

Query, StoredProc

Description

An attempt to close a query or stored procedure occurs when its *active* property, or the *active* property of the object's database, is set to *false*. If the object's rowset has been modified, *dBASE Plus* will try to save it, so the close attempt can be canceled by the rowset's *canSave* event handler. If not, the row is saved.

The close can also be prevented by the Query or StoredProc object's *canClose* event handler. If not, the object is closed, and its *onClose* event fires.

Because *onClose* fires after the rowset has closed, you can no longer affect its fields. If you want to do something with the rowset's data when the rowset closes, use the *canClose* event instead, and have the event handler return *true*.

onDelete

Topic group Related topics

Event fired after a row is successfully deleted.

Parameters

none

Property of

Rowset

Description

A row may be deleted explicitly by calling the *delete*( ) method, or implicitly via the user interface by choosing Delete Rows from the default Table menu or toolbar while editing table rows. While the *canDelete* fires first to determine if the row is actually deleted, *onDelete* fires after the row has been removed.

Because the row has been removed by the time *onDelete* fires, the row cursor is at the next row or the end-of-set when *onDelete* fires. However, this movement is not considered navigation, so if you have an *onNavigate* event handler, you should call it from *onDelete*.

onEdit

Topic group Related topics

Event fired after the rowset successfully enters Edit mode.

Parameters

none

Property of

Rowset

Description

The *beginEdit*( ) method is called (implicitly or explicity) to put the rowset in Edit mode. While the *canEdit* event fires first to see if the switch to Edit mode actually takes place, *onEdit* fires after the rowset has switched to Edit mode.

You can use *onEdit* to do things like automatically record when edits take place, or to save original values for auditing.

onGotValue

Topic group Related topics

Event fired after a field's *value* property is successfully read.

Parameters

none

Property of

Field (including DbfField, PdxField, SqlField)

Description

onGotValue is fired when reading a field's *value* property explicitly and when it is read to update a *dataLink*ed control. It does not fire when the field is accessed internally for SpeedFilters, index expressions, or master-detail links, or when calling *copyToFile*( ).

onNavigate

Topic group Related topics Example

Event fired after successful navigation in a rowset.

Parameters

<method expN>

Numeric value that indicates which method was called to fire the event:

Value	Method
1	next( )
2	first( )
3	last( )

4 All other navigation

<rows expN>

Number of rows *next*( ) method was called with. Zero if *next*( ) was not used.

Property of

Rowset

Description

Navigation in a rowset may occur explicitly by calling a navigation method like *next*( ) or *goto*( ), or implicitly via the user interface by choosing a navigation option from the default Table menu or toolbar while viewing a rowset. While *canNavigate* fires first before the row cursor has moved to see if the navigation actually takes place, *onNavigate* fires after the row position has settled on the desired row or end-of-set.

Because *onNavigate* fires when moving to the end-of-set and you cannot access field values when you're at the end-of-set, you may want to test the rowset's *endOfSet* property before you attempt to access field values in your *onNavigate* handler.

You can use *onNavigate* to update non-*dataLink*ed controls or calculated fields. In that case, you may want to call your *onNavigate* handler from the *onOpen* event as well, so that these objects are up-to-date when the rowset first opens.

When navigation occurs because a row has been abandoned or deleted, *onNavigate* does not fire. Call the *onNavigate* event handler from the *onAbandon* and *onDelete* event handler.

onOpen

Topic group Related topics Example

Event fired after query or stored procedure is opened successfully.

Parameters

none

Property of

Query, StoredProc

Description

onOpen fires after the Query or StoredProc object has successfully opened after its *active* property has been set to *true*.

onProgress

Topic group Example

Event fired periodically during long-running data processing operations.

Parameters

<percent expN>

The approximate percent-complete of the operation, from 0 to 100. When a message is passed, <percent expN> is the value -1.

<message expC>

A text message from the database engine.

Property of

Session

Description

Use *onProgress* to display progress information during data processing operations such as copying or indexing.

onProgress fires for the following operations:

Database::copyTable( )	COPY TABLE	All UpdateSet methods	APPEND FROM
Database::createIndex( )	COPY TO	INDEX ON	SORT

The *onProgress* event handler receives two parameters, but only one of them is valid for any given event. You may get either:

1. A percent-complete from 0 to 100 in <percent expN>, in which case <message expC> is a blank string, or
    - A message in <message expC>, in which case <percent expN> is -1.

onSave

Topic group Related topics

Event fired after successfully saving the row buffer.

Parameters

none

Property of

Rowset

Description

The row buffer may be saved explicitly by calling *save*( ), or implicitly by navigating in the rowset or closing the rowset. While *canSave* is fired first to verify that data is good before allowing it to be written, *onSave* fires after the row has been saved.

open( )

Topic group Related topics

Opens a database connection.

Syntax

This method is called implicitly by the Database object.

Property of

Database

Description

The *open*( ) method opens the database connection. It is called implicitly when you set the Database object's *active* property to *true*. In typical usage, you do not call this method directly.

Advanced applications may override the definition of this method to perform supplementary actions when opening the database connection. Custom data drivers must define this method to perform the appropriate actions to open their database connection.

packTable( )

Topic group Related topics

Packs a Standard table by removing all deleted rows.

Syntax

<oRef>.*packTable*(<table name expC>)

<oRef>

The database in which the table exists.

<table name expC>

The name of the table you want to pack.

Property of

Database

Description

For DBF (dBASE) tables, *packTable*( ) removes all the records in a table that have been marked as deleted, making all the remaining records contiguous. As a result, the records are assigned new record numbers and the disk space used is reduced to reflect the actual number of records in the table. Adding an autoincrement field will automatically pack the DBF table.

For DB (Paradox) tables, *packTable*( ) removes all deleted records and redistributes the remaining records in the record blocks, optimizing the block structure.

To refer to a Standard table, you can always use the default database in the default session by referring to it through the *databases* array property of the _app object. For example,

```
_app.databases[ 1 ].packTable( "Customer" )
```

A couple observations regarding the *packTable*( ) method:

The *packTable*( ) method returns *true* or *false* to indicate whether the packing operation was succesfull.

Packing is a maintenance operation that requires exclusive access to the table. The *packTable*( ) method will fail - return *false* - if someone else has the table open.

The *packTable*( ) method can return a value of *true* without any records actually being deleted. A return value of *true* only indicates the operation encountered no errors. It does not imply that records were deleted. If no records were marked as deleted, the *packTable*( ) method will return *true* as long as it does not encounter any errors.

In order to catch any errors that might occur, it is recommended that *packTable*( ) be used in a try/endtry construct.

params

Topic group Related topics Example

Parameters for an SQL statement or stored procedure call.

Property of

Query, StoredProc

Description

The *params* property contains an associative array that contains parameter names and values, if any, for an SQL statement in a Query object or a stored procedure call in a StoredProc object.

For a Query object, assigning an SQL statement with parameters to the *sql* property automatically creates the corresponding elements in the *params* array. Parameters are indicated by colons. The values you want to substitute are then assigned to the array elements in one of two ways:

1. Manually, before the query is activated or requeried with *requery*( ).
   - By assigning a *masterSource* to the query, in which case parameters are substituted with the matching fields from the *fields* array of the *masterSource*'s *rowset*. Parameters are matched to fields by name.

For a StoredProc object, the Borland Database Engine will try to get the names and types of any parameters needed by a stored procedure, once the procedure name is assigned to the *procedureName* property. This works to varying degrees for most SQL servers. If it succeeds, the *params* array is filled automatically with the corresponding Parameter objects. You must then assign the values you want to substitute to the *value* property of those objects.

For SQL servers that do not return the necessary stored procedure information, include the parameters, preceded with colons, in parentheses after the procedure name. The corresponding Parameter objects in the *params* array will be created for you; then you must assign the necessary *type* and *value* information.

picture

Topic group Related topics

A template that formats input to a DB (Paradox) field.

Property of

PdxField

Description

A *picture* uses special template symbols to format data entry into a field. However, many Paradox template symbols do not match *dBASE Plus* template symbols, so a *picture* for a DB field probably won't work as-is in the *picture* property of a control unless it's very simple, for example "999.99".

dBASE Plus does not enforce the DB field template. The *picture* property is informational only.

precision

Topic group Related topics

The number of digits allowed in an SQL-based field.

Property of

SqlField

Description

The *precision* property specifies the maximum number of digits that can be stored in a field represented by the SqlField object. The more digits allowed, the greater the precision or accuracy of a number.

prepare( )

Topic group Related topics Example

Prepares an SQL statement or stored procedure.

Syntax

<oRef>.*prepare*( )

<oRef>

The object you want to prepare.

Property of

Query, StoredProc

Description

prepare( ) prepares the stored procedure named in the *procedureName* property of a StoredProc object or the SQL statement stored in the *sql* property of a Query object. If the object is connected to an SQL-server-based database, the prepare message is passed on to the server.

Preparing an SQL statement or stored procedure call includes compiling the statement and setting up any optimizations. If the statement includes parameters, the statement can be prepared first, and, sometime later, you can get the parameter values from the client. Then the prepared statement and its parameters are ready for execution. By separating the client and server activities, things run a bit faster.

Preparing is part of the process that occurs when you set an object's *active* property to *true*, so you're never required to call *prepare*( ) explicitly.

problemTableName

Topic group Related topics

Name of the table in which you want to collect rows that could not be used during an update operation because of some problem other than a key violation.

Property of

UpdateSet

Description

In addition to key violations, problems during update operations are often caused by things like mismatched fields. If a row could not be transferred from the *source* to the *destination* because of a problem, it is instead copied to the table specified by the *problemTableName* property.

procedureName

Topic group Related topics Example

The name of the stored procedure to call.

Property of

StoredProc

Description

Set the *procedureName* property to the name of the procedure to call. The Borland Database Engine will try to get the names and types of any parameters needed by the stored procedure.

The following databases return complete parameter name and type information:

InterBase
Oracle
ODBC, if the particular ODBC driver provides it

The following databases return the parameter name but not the type:

Microsoft SQL Server
Sybase

The following database does not return any parameter information:

Informix

If the BDE can get the parameter names, the *params* array is filled automatically with the corresponding Parameter objects. You must then assign the values to substitute to the *value* property of those objects.

For SQL servers that do not return the necessary stored procedure information, include the parameters, preceded with colons, in parentheses after the procedure name. Empty Parameter objects will be created.

If the *type* of the parameter or the data type of the *value* for output parameters is not provided automatically, it must be set before calling the stored procedure, in addition to any input values.

readOnly

Topic group Related topics

Whether a DBF (dBASE) or DB (Paradox) field is read-only.

Property of

DbfField, PdxField

Description

readOnly indicates whether the field represented by the Field object is read-only or not.

ref

Topic group Related topics Example

A reference to the active data module object.

Property of

DataModRef

Description

After activating the DataModRef object, you may reference the data module object through the DataModRef object's *ref* property.

refresh( )

Topic group Related topics

Refreshes data in the entire rowset.

Syntax

<oRef>.*refresh*( )

<oRef>

The rowset you want to refresh.

Property of

Rowset

Description

To increase performance, rows are cached in memory as they are encountered. If the row cursor revisits a cached row, it can be reread quickly from memory instead of the disk. *refresh*( ) purges all cached rows—not to be confused with cached updates—for the rowset, forcing *dBASE Plus* to reread the data from disk. It discards any changes to the row buffer, so a row that has been modified is not saved. When the rowset is refreshed, any *dataLink*ed controls are also refreshed with values for the current row if *notifyControls* is *true*.

refresh( ) does not regenerate the rowset. If the rowset is not *live*, *refresh*( ) has no effect. Use *requery*( ) to regenerate the rowset.

refreshControls( )

Topic group Related topics

Refreshes any controls that are *dataLink*ed to the current row.

Syntax

<oRef>.*refreshControls*( )

<oRef>

The rowset you want to refresh.

Property of

Rowset

Description

refreshControls( ) updates any controls that are *dataLink*ed to Field objects in the rowset, regardless of the setting of the *notifyControls* property. The controls are updated with the values in the row buffer, not the values on disk.

Use *refreshRow*( ) first to refresh the fields in the row buffer with the values on disk if desired.

refreshRow( )

Topic group Related topics

Refreshes data in the current row.

Syntax

<oRef>.*refreshRow*( )

<oRef>

The rowset you want to refresh.

Property of

Rowset

Description

refreshRow( ) rereads the data for the current row from disk. It discards any changes to the row buffer, so a row that has been modified is not saved. When the row is refreshed, any *dataLink*ed controls are also refreshed if *notifyControls* is *true*.

Use *refresh*( ) to refresh the entire rowset.

reindex( )

Topic group

Rebuilds a Standard table's indexes from scratch.

Syntax

<oRef>.*reindex*(<table name expC>)

<oRef>

The database in which the table exists.

<table name expC>

The name of the table you want to reindex.

Property of

Database

Description

Indexes can become unbalanced during normal use. Occasionally, they can also be corrupted. In both cases, you can fix the problem by using *reindex*( ), which rebuilds the indexes from scratch.

Reindexing is a maintenance operation and requires exclusive access to the table; no one else may have it open at the time, or *reindex*( ) will fail.

To refer to a Standard table, you can always use the default database in the default session by referring to it through the *databases* array property of the _app object. For example,

```
_app.databases[ 1 ].reindex( "Customer" )
```

renameTable( )

Topic group Related topics

Renames a table in a database.

Syntax

<oRef>.*renameTable*(<old name expC>, <new name expC>)

<oRef>

The database in which to rename the table.

<old name expC>

The current name of the table.

<new name expC>

The new name of the table.

Property of

Database

Description

renameTable( ) renames a table in a database, including all secondary files such as index and memo files.

The table to rename should not be open.

By specifying a path in <new name expC>, the table, together with its' associated files, is moved to that destination and renamed <new name expC>. Associated files are moved regardless of whether <old name expC> uses the .dbf designation.

If a path is specified in <old name expC>, and no path is specified in <new name expC>, the table is moved to the location of the <oRef> database or (in the case of the default database_app.databases[1]) to the default directory.

To rename a Standard table, you can always use the default database in the default session by referring to it through the *databases* array property of the _app object. For example,

```
_app.databases[ 1 ].renameTable( "Before", "After" )
```

replaceFromFile( )

Topic group Related topics Example

Copies the contents of a file into a BLOB field.

Syntax

<oRef>.*replaceFromFile*(<file name expC> [,<append expL>])

<oRef>

The BLOB field you want to copy into.

<file name expC>

The name of the file you want to copy.

<append expL>

Whether to append the new data or overwrite.

Property of

Field

Description

replaceFromFile( ) copies the contents of the named file into the specified BLOB field.

By specifying <*append expL*> as *true*, the contents of the file are added to the end of the current contents of the BLOB field. If the parameter is specified as *false* or left out, the BLOB field will be overwritten and end up containing only the contents of the file.

If you don't include an extension for <*file name expC*>, *dBASE Plus* assigns a .TXT extension. If you don't wish to pass a file extension, follow <*file name expC*> with a dot.

```
myfilename.
```

requery( )

Topic group Related topics Example

Re-executes the query or stored procedure, regenerating the rowset.

Syntax

<oRef>.*requery*( )

<oRef>

The query or stored procedure you want to re-execute.

Property of

Query, StoredProc

Description

requery( ) re-executes a stored procedure or a query's SQL statement, generating an up-to-date rowset. Calling *requery*( ) is similar to setting the object's *active* property to *false* and back to *true*, except that *requery*( ) does not prepare the SQL statement. This includes attempting to save the current row if necessary and closing the object, firing all the events along the way. If those actions are halted by the *canSave* or *canClose* event handlers, the *requery*( ) attempt will stop at that point.

Use *requery*( ) when a parameter in the SQL statement has changed to re-execute the query with the new value.

Use *refresh*( ) to refresh the rowset without re-executing the query, which is faster. But *refresh*( ) has no effect on a rowset that is not *live*; use *requery*( ) instead.

requestLive

Topic group Related topics

Specifies whether the query should generate an editable rowset.

Property of

Query

Description

Before making a query active, you can determine whether the rowset that is generated is editable or not. You can choose to make it not editable to prevent accidental modification of the data.

requestLive defaults to *true*.

required

Topic group Related topics

Whether a field is required to be filled in and not left blank.

Property of

DbfField, PdxField

Description

required indicates whether the field represented by the Field object is a required field; that is, whether it must be filled in.

rollback( )

Topic group Related topics

Cancels the transaction by undoing all logged changes

Syntax

<oRef>.*rollback*( )

<oRef>

The database whose changes you want to rollback.

Property of

Database

Description

A transaction works by logging all changes. If an error occurs while attempting one of the changes, or the changes need to be undone for some other reason, the transaction is canceled by calling the *rollback*( ) method. Otherwise, *commit*( ) is called to clear the transaction log, thereby indicating that all the changes in the transaction were committed and that the transaction as a whole was posted.

Since new rows have already been written to disk, rows that were added during the transaction are deleted. In the case of DBF (dBASE) tables, the rows are marked as deleted, but are not physically removed from the table. If you want to actually remove them, you can pack the table with *packTable*( ). Rows that were just edited are returned to their saved values.

All locks made during a transaction are maintained until the transaction is completed. This ensures that no one else can make any changes until the transaction is committed or abandoned.

rowCount( )

Topic group Related topics

Returns the logical row count.

Syntax

<oRef>.*rowcount*( )

<oRef>

The rowset you want to count.

Property of

Rowset

Description

rowCount( ) returns the logical row count of the rowset, if known. The logical row count is the number of rows in the rowset, using the rowset's current index and filter conditions.

Determining the logical row count is often an expensive operation, requiring that the rows actually be counted individually. When the count is not known, *rowCount*( ) returns the value -1;

it does not attempt to get the count. If your application requires the actual row count, use the *count*( ) method to count the rows if *rowCount*( ) returns -1.

Note

rowCount( ) is different from the function RECCOUNT( ). RECCOUNT( ) returns the number of physical records in a table. *rowCount*( ) returns the logical count in a rowset. These numbers are not guaranteed to be the same, even with a SELECT * query of a DBF table, because *rowCount*( ) must consider deleted records—it does not know if there are any unless it actually looks—while RECCOUNT( ) does not.

rowNo( )

Topic group Related topics

Returns the current logical row number in the rowset.

Syntax

<oRef>.*rowno*( )

<oRef>

The rowset containing the current row.

Property of

Rowset

Description

rowNo( ) returns the current logical row number in the rowset, if known. The logical row number is the relative row number, using the rowset's current index and filter conditions. The first row is row number 1 and the last row is equal to the number of rows in the current rowset.

In some cases, for example scrolling with the scrollbar in a grid to an arbitrary location and clicking on a row, the logical row number is not known, and would have to be calculated. In contrast, if you were to page down repeatedly to that same location, the row number is known, because it is updated as you move from page to page in the grid. When the row number is not known, *rowNo*( ) returns the value -1.

Note

rowNo( ) is different from the function RECNO( ). RECNO( ) returns the physical record number of the current row in a DBF table, which never changes (unless the table is PACKed). *rowNo*( ) returns the logical row number; the same physical record will have a different logical row number, depending on the current index and filter.

rowset

Topic group Related topics

A reference to the query's or stored procedure's rowset, or a data module's primary rowset.

Property of

DataModule, Query, StoredProc

Description

A Query object always contains a *rowset* property, but that property does not refer to a valid Rowset object until the query has been activated and the rowset has been opened.

Some stored procedures generate rowsets. If that is the case, the StoredProc object's *rowset* property refers to that rowset after the stored procedure is executed.

A data module may designate a primary rowset. This rowset is assigned to a form's *rowset* property by the Form designer when the data module is used in the form.

The *rowset* property is read-only for Query and StoredProc objects.

For more information, see class Rowset.

save( )

Topic group

Saves the current row buffer.

Syntax

<oRef>.*save*( )

<oRef>

The rowset you want to save.

Property of

Rowset

Description

After a row has been modified, you must call *save*( ) to write the row buffer to a rowset or table. By design, *save*( ) has no effect if the rowset's *modified* property is *false*, because supposedly there are no changes to save; and a successful *save*( ) sets the *modified* property to *false*, indicating that values in the controls do not differ from those on the disk. You can manipulate the *modified* property to control this designed behavior.

The *canSave* event fires after calling *save*( ). If there is no *canSave* event handler, or *canSave* returns *true*, then the row buffer is saved, the *modified* property is set to *false*, and the *onSave* event fires.

The row cursor does not move after a *save*( ) unless the values saved cause the row to become out-of-set. In that instance, the row cursor is moved to the next available row or, if there are no more available rows, the end-of-set .

Changes are written to disk unless the *cacheUpdates* property is set to *true*, in which case the changes are cached. Whether the changes are actually written to a physical disk depends on the operating system and its own disk caches, if any.

scale

Topic group Related topics

The number of digits, to the right of the decimal point, that can be stored in an SQL-based field

Property of

SqlField

Description

The *scale* property specifies the number of digits, to the right of the decimal point, that can be stored in the SqlField object

session

Topic group Related topics

The Session object to which the database, query, or stored procedure is assigned.

Property of

Database, Query, StoredProc

Description

A database must be assigned to a session. When a Database object is created, it's automatically assigned to the default session.

A query or stored procedure must be assigned to a database, which in turn is assigned to a session. When created, a Query or StoredProc object is assigned to the default database in the default session.

To assign either the Query, or StoredProc, object to the default database in another session, assign that session to their *session* property. Assigning a Query or StoredProc's *session* property always sets their *database* property to the default database in that session.

To assign either the Query, or StoredProc, object to another database in another session, assign the object to that session first. This makes the databases in that session available to the object.

To enable the Session object's security features, the database the Session object is assigned to must be active.

setRange( )

Topic group Related topics

Constrains the rowset to those rows whose key field values falls within a range.

Syntax

<oRef>.*setRange*(<key exp>)

or

<oRef>.*setRange*(<startKey exp> | *null*, <endKey exp> | *null*)

<oRef>

The rowset you want to constrain.

<key exp>

Shows only those rows whose key value matches <key exp>.

<startKey exp>

Shows those rows whose key value is equal to or greater than <startKey exp>.

<endKey exp>

Shows those rows whose key value is less than or equal to <endKey exp>.

There are four ways to use *setRange*( ):

1. Exact match: *setRange*(<key exp>)
    - Range from start to end: *setRange*(<startKey exp>, <endKey exp>)
    - Range from starting value: *setRange*(<startKey exp>, *null*)

- Range up to ending value: *setRange*(*null*, <endKey exp>)

Property of

Rowset

Description

setRange( ) is similar to a filter; *setRange*( ) uses the rowset's current index (represented by its *indexName* property) and shows only those rows whose key value matches a single value or falls within a range of values. This is referred to as a key constraint. Because it uses an index, a key constraint is instantaneous, while a filter condition must be evaluated for each row. Use *clearRange*( ) to remove the constraint.

The key range values must match the key expression of the index. For example, if the index key is UPPER(Name), specify uppercase letters in the range expressions. For character expressions, the key match is always a partial string match (starting at the beginning of the expression); therefore, an exact match with <key exp> could match multiple key values if the <key exp> is shorter than the key expression.

When you use both *setRange*( ) and a filter (and *canGetRow*) for the same rowset, you get those rows that are within the index range and that also meet the filter condition(s).

Rowsets that use *masterRowset* for master-detail linkage internally apply *setRange*( ) in the detail rowset. If you use *setRange*( ) in the detail rowset, it overrides the master-detail key constraint. Navigation in the master rowset would reapply the master-detail constraint.

share

Topic group

How to share data access resources.

Property of

Database, DataModRef

Description

The *share* property controls how database connections and dataModules are shared. The *share* property is an enumerated property that can be assigned one of the following:

Value	Description
0	None
1	All

Database objects

Multiple Database objects may share the same database connection. Sharing database connections reduces resource usage on both the client and server. Some servers have a maximum number of simultaneous connections, so sharing connections will also allow more users to connect to the server.

When set to All (the default), all Database objects with the same *databaseName* property (running in the same instance of *dBASE Plus*) will share the same database connection. When set to None, each Database object will use its own connection.

DataModref objects

When set to All, all DataModRef objects with the same *dataModClass* property will share the same instance of that class; the same DataModule object. This means, for example, that navigation performed by one user of the DataModRef is seen by all users of that same *dataModClass* (if their *share* property is also set to All). dataModule sharing is only useful in limited cases. For typical usage, the *share* property should be set to None, the default.

source

Topic group Related topics

The source rowset or table of an UpdateSet operation.

Property of

UpdateSet

Description

The *source* property contains an object reference to a rowset or the name of a table that is the source of an UpdateSet operation. For an *append*( ), *update*( ), or *appendUpdate*( ), it refers to the rowset or table that contains the new data. For a *copy*( ), it refers to the rowset or table that is to be duplicated. For a *delete*( ), the *source* property refers to the table that contains the list of rows to be deleted.

The *destination* property specifies the other end of the UpdateSet operation.

sql

Topic group Related topics Example

The SQL statement that describes the query.

Property of

Query

Description

The *sql* property of a Query object contains an SQL SELECT statement that describes the rowset to be generated. To use a stored procedure in an SQL server that returns a rowset, use the *procedureName* property of a StoredProc object instead.

The *sql* property must be assigned before the Query object is activated.

The SQL SELECT statement may contain an ORDER BY clause to set the row order, a WHERE clause to select a subset of rows, perform a JOIN, or any other SQL SELECT clause.

But to take full advantage of the data objects' features—such as locating and filtering—with SQL-server-based tables, the SQL SELECT used to access a table must be a simple SELECT: all the fields from a single table, with no options. For example,

```
select * from CUSTOMER
```

If the SQL statement is not a simple SELECT, locating and filtering is performed locally, instead of by the SQL server. If the result of the SELECT is a small rowset, local searching will be fast; but if the result is a large rowset, local searching will be slow. For large rowsets, you should use a simple SELECT, or use parameters in the SQL statement and *requery*( ) as needed instead of relying on the Locate and Filter features.

Master-detail linking through the *masterRowset* and *masterFields* properties with SQL-server-based tables also requires a simple SELECT. An alternative is master-detail linking though Query objects with the *masterSource* property and parameters in the SQL statement. There is no simple SELECT restriction when using Standard tables.

Parameters in an SQL statement are indicated by a colon. For example,

```
select * from CUST where CUST_ID = :cust_id
```

Whenever the SQL property is assigned, it is scanned for parameters. *dBASE Plus* automatically creates corresponding elements in the query's *params* array, with the name of the parameter as the array index. For more information, see the *params* property.

In addition to assigning the SQL statement directly to the *sql* property, you may also use an SQL statement in an external file. To use an external file, place an "@" symbol before the file name in the *sql* property. For example,

```
@ORDERS.SQL
```

The external file must be a text file that contains an SQL statement.

state

Topic group Related topics Example

An enumerated value indicating the rowset's current mode.

Property of

Rowset

Description

The *state* property is read-only, indicating which mode the rowset is in, as listed in the following table:

Value	Mode
0	Closed
1	Browse
2	Edit
3	Append
4	Filter
5	Locate

When the rowset's query is not active, the rowset is Closed.

While the query is active, the rowset is in Browse mode when it's not in one of the next four modes.

The rowset is in Edit mode after a successful *beginEdit*( ) (implicit or explicit) and it stays in that mode until the row is saved or abandoned.

After a successful *beginAppend*( ), it is in Append mode. It stays in that mode until the new row is saved or abandoned.

After a *beginFilter*( ), it is in Filter mode. It stays in that mode until there is an *applyFilter*( ) or the Filter mode is abandoned.

After a *beginLocate*( ), it is in Locate mode. It stays in that mode until there is an *applyLocate*( ) or the Locate mode is abandoned.

tableDriver

Topic group Related topics

The Driver currently being used to access a table

Property of

Rowset

Description

Returns a character string indicating the driver currently being used. For example, "dBASE", "FOXPRO" or "PARADOX" for native local tables; "Advantage 32 bit" for the Advantage ODBC 32 bit driver; "ORACLE", "INTERBASE " or "MS SQL" for a few of the SQL Link drivers.

Read-only.

tableExists( )

Topic group

Checks to see if a specified table exists in a database.

Syntax

<oRef>.*tableExists*(<table name expC>)

<oRef>

The database in which to see if the table exists.

<table name expC>

The name of the table you want to look for.

Property of

Database

Description

tableExists( ) returns *true* if a table with the specified name exists in the database.

To look for a Standard table, you can always use the default database in the default session by referring to it through the *databases* array property of the _app object. For example,

```
_app.databases[ 1 ].tableExists( "Billing" )
```

If you do not specify an extension, *dBASE Plus* will look for both a DBF (dBASE) and DB (Paradox) table with that name.

tableLevel

Topic group Related topics

The version of the current local table

Property of

Rowset

Description

Returns an integer indicating the version of the current local table. Currently only the BDE's dBASE, FoxPro and Paradox provide a non-zero value for this property. *tableLevel* values include; 3 for dBASE III, 4 for dBASE IV, 5 for dBASE 5 (when containing OLE or BINARY fields), 7 for Vdb7, 25 for FoxPro 2.5 and 5 for Paradox.

Read-only

tableName

Topic group Related topics

The current table name

Property of

Rowset, TableDef

Description

Returns a character string indicating the name of the table a current rowset is based on.

Read-only.

tempTable

Topic group Related topics

A status of the current table

Property of

Rowset

Description

Returns a logical (True/.T.) when the current table (referenced by *tableName* is a temporary table.

Read-only.

type [Field]

Topic group Related topics

The data type of the value stored in a field.

Property of

Field (including DbfField, PdxField, SqlField)

Description

The *type* property reflects the data type stored in the field represented by the Field object. For a list of data types, see *Field types.*

type [Parameter]

Topic group Related topics

An enumerated value indicating the type of parameter.

Property of

Parameter

Description

The *type* property indicates the type of parameter a Parameter object represents, as listed in the following table:

Value	Type
0	Input
1	Output
2	InputOutput
3	Result

See the Parameter object's *value* property for details on each type.

unidirectional

Topic group Related topics

Specifies whether to assume forward-only navigation to increase performance on SQL-based servers.

Property of

Query

Description

If *unidirectional* is set to *true*, previously visited rows are not cached and less communication is required between *dBASE Plus* and the SQL server. This results in fewer resources consumed and better performance, but is worthwhile only if you never want to go backward in the rowset.

If *unidirectional* is *true*, you may still be able to go backward, depending on the server, but if so it would be time-consuming.

unique

Topic group

Prevents multiple records with the same expression value from being included in the index. Includes only the first record for each value.

Property of

DbfIndex, Index

Description

Restricts the row included in the index to those with unique key values. If two or more rows have the same key value, only the first row with the key value is included. The "first" row is determined by row number (the order in which you entered rows). This field does not apply to secondary indexes on Paradox tables. A primary index in a Paradox table requires keys to be unique.

unlock( )

Topic group Related topics Example

Releases row and rowset locks.

Syntax

<oRef>.*unlock*( )

<oRef>

The rowset that contains the lock.

Property of

Rowset

Description

unlock( ) releases automatic row locks and locks set by *lockRow*( ) and *lockSet*( ).

You cannot release locks during a transaction.

unprepare( )

Topic group Related topics

Releases the server resources used by a query or stored procedure.

Syntax

This method is called implicitly by the Query or StoredProc object.

Property of

Query, StoredProc

Description

The *unprepare*( ) method cleans up after a query or stored procedure is deactivated. It is called implicitly when you set the object's *active* property to *false*. In typical usage, you do not call this method directly.

Advanced applications may override the definition of this method to perform supplementary actions when deactivating the query or stored procedure. Custom data drivers must define this method to perform any necessary actions to clean up when a query or stored procedure is deactivated.

update

Topic group Related topics Example

The date and time of the last update made to the row.

Property of

LockField

Description

Use *update* to determine the date and time the row or table was last updated. The date and time are represented in a string in the following format:

```
MM/DD/YY HH:MM:SS
```

This format is accepted by the constructor for a Date object, so you can easily convert the *update* string into an actual date/time.

This property is available only for DBF tables that have been CONVERTed.

update( )

Topic group Related topics

Updates existing rows in one rowset from another.

Syntax

<oRef>.*update*( )

<oRef>

The UpdateSet object that describes the update.

Property of

UpdateSet

Description

Use *update*( ) to update a rowset. You must specify the UpdateSet object's *indexName* property that will be used to match the records. The index must exist for the destination rowset. The original values of all changed records will be copied to the table specified by the UpdateSet object's *changedTableName* property.

To add new rows and update existing rows only, use the *appendUpdate*( ) method instead.

updateWhere

Topic group Related topics

Determines which fields to use in constructing the WHERE clause in an SQL UPDATE statement. SQL-based servers only.

Property of

Query

Description

updateWhere is an enumerated property that may be one of the following values:

Value	Description
0	All fields
1	Key fields
2	Key fields and changed fields

usePassThrough

Topic group

Controls whether or not a query, with a simple sql select statement (of the form "select * from <table>), is sent directly to the DBMS for execution or is setup to behave like a local database table.

Property of

Query

Description

When the *usePassThrough* property is set to False (the default):

For query's using a simple sql select statement (in the form "select * from table") which meet the conditions listed below in "Conditions for Dynamic Caching", the query's rowset, when activated, is setup to behave like a local, file based database table such as a dBASE .dbf table.

The query result set is managed using a dynamic caching algorithm, as described below in Dynamic Caching Behavior", which supports the use of index and key-oriented operations.

Query's using a complex sql select statement, or those which do not meet the conditions described below in Conditions for Dynamic Caching, will be executed as if the usePassThrough property were set to True.

When the *usePassThrough* property is set to True, the query's sql statement, is passed directly through to the database server for execution and the resulting rowset uses a more basic caching algorithm, described below in "Basic Caching Behavior". The query result set cannot use most index, or key oriented operations.

Conditions for Dynamic Caching

Query's *sql* property must contain a simple select statement ("select * from table").

The database server must support row ID's and/or the table must have a unique or primary key index defined.

Dynamic Caching Behavior

When opening a table to use dynamic caching:

The fastest index is chosen automatically if none was specified during table open.

A partial cache is kept, ordered by index.

The cache contains the current cursor row, plus the last several rows fetched.

The cache is automatically refreshed ,with up-to-date data, when row navigation occurs and can be manually refreshed by calling the rowset's *refresh*( ) method.

The order in which a table can be navigated may be set via the rowset's *indexName* property.

Key-oriented operations, such as *findKey*( ) and *setRange*( ), can be used.

Basic Caching Behavior

Basic caching is used if:

The conditions for dynamic caching are not met
or
The *usePassThrough* property is set to False

With basic caching:

Every row fetched is cached on the workstation in case it is needed again.

The cache is not automatically refreshed. To refresh the cache you must call the query's *requery*( ) method or re-execute the query by setting the query's *active* property to False and then back to True.

The order rows are navigated must be set via the sql select statement's ORDER BY clause, rather than via the rowset's *indexName* property.

Key-oriented operations such as *findKey*( ) and *setRange*( ) are not available.

However, bookmarks can be used as long as rows can be uniquely identified.

Pros and Cons of Dynamic Caching

Dynamic caching works well with tables of up to a few million rows.

Larger tables may take a considerable amount of time to open.

Pros and Cons of Basic Caching

Basic caching can be used to quickly retrieve initial results from queries on large tables (tables with more than a few million rows) as long as no ORDER BY clause is included in the sql statement. However, you still need to be careful to limit the number of rows retrieved to the workstation, as every row retrieved is cached in workstation memory and can quickly use up available memory if the result set is more than a few million rows in size.

user

Topic group Related topics Example

The name of the user that last locked or updated the row.

Property of

LockField

Description

Use *user* to determine the username of the person that currently has a lock when a lock attempt fails, or the name of the user that last had a lock on the row. The maximum length of *user* depends on the size of the _DBASELOCK field specified when the table was CONVERTed.

This property is available only for DBF tables that have been CONVERTed.

user( )

Topic group Related topics

Returns the login name of the user currently logged in to the session.

Syntax

<oRef>.*user*( )

<oRef>

The session you want to check.

Property of

Session

Description

user( ) returns the login name of the user currently logged in to a session on a system that has DBF table security in place. If no DBF table security has been configured, or no one has logged in to the session, *user*( ) returns an empty string.

value [Field]

Topic group Related topics Example

The value of a field in the row buffer.

Property of

Field (including DbfField, PdxField, SqlField)

Description

All of the Field objects in the rowset's *fields* array property have a *value* property, which reflects the value of the field in the row buffer, which in turn reflects the values of the fields in the current row.

You may attempt to change the value of a *value* property directly by assignment, in which case the attempt occurs immediately, or through a *dataLink*ed control, in which case the attempt occurs when the control loses focus. In either case, the field's *canChange* property fires to see

whether the change is allowed. If *canChange* returns *false*, then the assignment doesn't take; if the change was through a *dataLink*ed control, the control still contains the proposed new value. If *canChange* returns *true* or there is no *canChange* event handler, the field's value is changed and the *onChange* event fires.

When a field is changed, the rowset's *modified* property is automatically set to *true* to indicate that the rowset has been changed.

By using a field's *beforeGetValue* event, you can make the *value* property appear to be something else besides what is in the row buffer.

value [Parameter]

Topic group Related topics Example

The input, output, or result value of a stored procedure.

Property of

Parameter

Description

Values are transmitted to and from stored procedures through Parameter objects. Each object's *type* property indicates what type of parameter the object represents. Depending on which one of the four types the parameter is, its *value* property is handled differently.

Input: an input value for the stored procedure. The *value* must be set before the stored procedure is called.

Output: an output value from the stored procedure. The *value* must be set to the correct data type before the stored procedure is called; any dummy value may be used. Calling the stored procedure sets the *value* property to the output value.

InputOutput: both input and output. The *value* must be set before the stored procedure is called. Calling the stored procedure updates the *value* property with the output value.

Result: the result value of the stored procedure. In this case, the stored procedure acts like a function, returning a single result value, instead of updating parameters that are passed to it. Otherwise, the *value* is treated like an output value. The name of the Result parameter is always "Result".

If a Parameter object is assigned as the *dataLink* of a component in a form, changes to the component are reflected in the *value* property of the Parameter object, and updates to the *value* property of the Parameter object are displayed in the component.

version

Topic group Related topics

The version of the current local table

Property of

TableDef

Description

Returns an integer indicating the version of the current local table. Currently only the BDE's dBASE, FoxPro and Paradox provide a non-zero value for this property. These values include; 3 for dBASE III, 4 for dBASE IV, 5 for dBASE 5 (when containing OLE or BINARY fields), 7 for Vdb7, 25 for FoxPro 2.5 and 5 for Paradox.

The *version* property is Read-only

Form Objects

Form objects

Topic group

Forms are the primary visual components in *dBASE Plus* applications. You can create forms visually through the Form wizard or Form designer, or programatically by writing code and saving your work as a .WFM file.

Common visual component properties

Topic group

These properties, events, and methods are common to many visual form components:

Property	Default	Description
before		The next object in the z-order
borderStyle	Default	Specifies whether a box border appears (0=Default, 1=Raised, 2=Lowered, 3=None, 4=Single, 5=Double, 6-Drop Shadow, 7=Client, 8=Modal, 9=Etched In, 10=Etched Out)
dragEffect	0	The type of Drag&Drop operation to be performed (0=None, 1=Copy, 2=Move)
enabled	true	Whether a component can get focus and operate
fontBold	false	Whether the text in a component appears in bold face
fontItalic	false	Whether the text in a component appears italicized
fontName	Arial	The typeface of the text in a component
fontSize	10	The point size of the text in a component
fontStrikeout	false	Whether the text in a component appears striked-through
fontUnderline	false	Whether the text in a component is displayed underlined
form		The form that contains a component
height		Height in the form's current *metric* units
helpFile		Help file name
helpId		Help index topic or context number for context-sensitive help
hWnd		The Windows handle for a component
id	-1	Supplementary control ID number
left	0	The location of the left edge of a component in the form's current *metric* units, relative to the left edge of its container
mousePointer	0	The mouse pointer type when the pointer is over a component
name		The name of a component
pageno	1	The page of the form on which a component appears
parent		A component's immediate container
printable	true	Whether a component is printed when the form is printed
speedTip		Tool tip displayed when pointer hovers over a component
statusMessage		Message displayed in status bar when a component has focus
tabStop	true	Whether a component is in the tab sequence

top	0	The location of the top edge of a component in the form's current *metric* units, relative to the top edge of its container
visible	true	Whether a component is visible
width		Width in the form's current *metric* units

Event	Parameters	Description
canRender		Reports only: before a component is rendered; return value determines whether component is rendered
onDesignOpen	<from palette expL>	After a component is first added from the palette and then every time the form is opened in the Form Designer
onDragBegin		When a Drag&Drop operation begins for a component
onGotFocus		After a component gains focus
onHelp		When F1 is pressed—overrides context-sensitive help
onLeftDblClick	<flags expN>, <column expN>, <row expN>	When the left mouse button is double-clicked
onLeftMouseDown	<flags expN>, <column expN>, <row expN>	When the left mouse button is pressed
onLeftMouseUp	<flags expN>, <column expN>, <row expN>	When the left mouse button is released
onLostFocus		After a component loses focus
onMiddleDblClick	<flags expN>, <column expN>, <row expN>	When the middle mouse button is double-clicked
onMiddleMouseDown	<flags expN>, <column expN>, <row expN>	When the middle mouse button is pressed
onMiddleMouseUp	<flags expN>, <column expN>, <row expN>	When the middle mouse button is released
onMouseMove	<flags expN>, <column expN>, <row expN>	When the is moved over a component
onOpen		After the form containing a component is opened
onRender		Reports only: after a component is rendered
onRightDblClick	<flags expN>, <column expN>, <row expN>	When the right mouse button is double-clicked
onRightMouseDown	<flags expN>, <column expN>, <row expN>	When the right mouse button is pressed
onRightMouseUp	<flags expN>, <column expN>, <row expN>	When the right mouse button is released
when	<form open expL>	When attempting to give focus to a component; return value determines whether the component gets focus.

Method	Parameters	Description
drag( )	<type expC>	Initiates a Drag&Drop operation for a component

	<name expC> <icon expC>	
move( )	<left expN> [,<top expN> [,<width expN> [,<height expN>]]]	Repositions and/or resizes a component
release( )		Explicitly releases a component from memory
setFocus( )		Sets focus to a component

class ActiveX

Topic group Related topics

Representation of an ActiveX control.

Syntax

[<oRef> =] new ActiveX(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created ActiveX object.

<container>

The container—typically a Form object—to which you're binding the ActiveX object.

<name expC>

An optional name for the ActiveX object. If not specified, the ActiveX class will auto-generate a name for the object.

Properties

The following table lists the properties of interest in the ActiveX class. (No particular events or methods are associated with this class.)

Property	Default	Description
anchor	0 – None	How the ActiveX object is anchored in its container (0=None, 1=Bottom, 2=Top, 3=Left, 4=Right, 5=Center, 6=Container)
baseClassName	ACTIVEX	Identifies the object as an instance of the ActiveX class
classId		The ID string that identifies the ActiveX control
className	(ACTIVEX)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
description		A short description of the ActiveX control
nativeObject		The object that contains the ActiveX control's own properties, events, and methods

The following table lists the common properties, events, and methods of the ActiveX class:

Property		Event		Method
before borderStyle dragEffect form height	pageno parent printable speedTip systemTheme	beforeRelease onClose onDragBegin onLeftDblClick onLeftMouseDown	onMouseMove onRightDblClick onRightMouseDown onRightMouseUp	drag( ) move( ) release( ) setFocus( )

left
name
top
width
onLeftMouseUp
onMiddleDblClick
onMiddleMouseDown
onMiddleMouseUp

Description

An ActiveX object in *dBASE Plus* is a place holder for an ActiveX control, not an actual ActiveX control.

To include an ActiveX control in a form, create an ActiveX object on the form. Set the *classId* property to the component's ID string.Once the *classId* is set, the component inherits all the published properties, events, and methods of the ActiveX control, which are accessible through the *nativeObject* property. The object can be used just like a native *dBASE Plus* component.

class Browse

Topic group Related topics

A data-editing tool that displays multiple records in row-and-column format.

Syntax

[<oRef> =] new Browse(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created Browse object.

<container>

The container—typically a Form object—to which you're binding the Browse object.

<name expC>

An optional name for the Browse object. If not specified, the Browse class will auto-generate a name for the object.

Properties

The following tables list the properties, events, and methods of interest in the Browse class.

Property	Default	Description
alias		The table that is accessed
allowDrop	false	Whether dragged objects (normally a table or table field) can be dropped in the browse object
anchor	0 – None	How the Browse object is anchored in its container (0=None, 1=Bottom, 2=Top, 3=Left, 4=Right, 5=Center, 6=Container)
append	true	Whether rows can be added
baseClassName	BROWSE	Identifies the object as an instance of the Browse class
className	(BROWSE)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorHighlight	WindowText /Window	The color of the highlighted cell
colorNormal	WindowText /Window	The color of all other cells
cuaTab	true	Whether pressing Tab follows CUA behavior and moves to next control, or moves to next cell

fields		The fields to display, and the options to apply to each field
frozenColumn		The name of the column inside which the cursor is confined.
lockedColumns	0	The number of columns that remain locked on the left side of the browse grid as it is scrolled horizontally.
modify	true	Whether the user can alter data
scrollBar	Auto	When a scroll bar appears for the Browse object (0=Off, 1=On, 2=Auto, 3=Disabled)

Event	Parameters	Description
onAppend		After a record is added to the table
onChange		After the user changes a value
onDragEnter	<left expN> <top expN> <type expC> <name expC>	When the mouse enters the Browse display area during a Drag&Drop operation
onDragLeave		When the mouse leaves the Browse display area without having dropped an object
onDragOver	<left expN> <top expN> <type expC> <name expC>	While the mouse drags an object over the Browse display area during a Drag&Drop operation
onDrop	<left expN> <top expN> <type expC> <name expC>	When the mouse button is released over the Browse display area during a Drag&Drop operation
onNavigate		After the user moves to a different record

Method	Parameters	Description
copy( )		Copies selected text to the Windows Clipboard
cut( )		Cuts selected text and to the Windows Clipboard
keyboard( )	<expC>	Simulates typed user input to the Browse object
paste( )		Copies text from the Windows clipboard to the current cursor position
undo( )		Reverses the effects of the most recent *cut*( ), *copy*( ), or *paste*( ) action

The following table lists the common properties, events, and methods of the Browse class:

Property		Event		Method
before	id	beforeRelease	onMiddleMouseDown	drag( )
borderStyle	left	onClose	onMiddleMouseUp	move( )
dragEffect	mousePointer	onDesignOpen	onMouseMove	release( )
enabled	name	onDragBegin	onOpen	setFocus( )
fontBold	pageno	onGotFocus	onRightDblClick	
fontItalic	parent	onHelp	onRightMouseDown	
fontName	printable	onLeftDblClick	onRightMouseUp	
fontSize	speedTip	onLeftMouseDown	when	
form	statusMessage	onLeftMouseUp		
height	systemTheme	onLostFocus		
helpFile	tabStop	onMiddleDblClick		
helpId	top			
hWnd	visible			
	width			

Description

The Browse object is maintained for compatibility and is suitable only for viewing and editing tables open in work areas. For forms that use data objects, use a Grid object instead.

Two properties specify which table is displayed in the Browse object.

The *view* property of the parent form
The *alias* property of the browse object

You can specify individual fields to display with the *fields* property. For example, if the browse object's form is based on a query, you use *fields* to display fields from any of the query's tables. (You must specify a file with *alias* before you can use *fields*.)

class CheckBox

Topic group Related topics

A check box on a form.

Syntax

[<oRef> =] new CheckBox(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created CheckBox object.

<container>

The container—typically a Form object—to which you're binding the CheckBox object.

<name expC>

An optional name for the CheckBox object. If not specified, the CheckBox class will auto-generate a name for the object.

Properties

The following tables list the properties and events of interest in the CheckBox class. (No particular methods are associated with this class.)

Property	Default	Description
baseClassName	CHECKBOX	Identifies the object as an instance of the CheckBox class
className	(CHECKBOX)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorNormal	BtnText/BtnFace	The color of the checkbox's text label
dataLink		The Field object that is linked to the CheckBox
group		The group to which the check box belongs
text	<same as name>	The text label that appears beside the check box
textLeft	false	Whether the check box's text label appears to the left or to the right of the check box
transparent	false	Whether the CheckBox object has the same background color or image as its container
value		The current value of the check box (*true* or *false*)

Event	Parameters	Description
onChange		After the check box is toggled

The following table lists the common properties, events, and methods of the CheckBox class:

Property		Event		Method
before	id	beforeRelease	onMiddleMouseDown	drag( )
borderStyle	left	onClose	onMiddleMouseUp	move( )
dragEffect	mousePointer	onDesignOpen	onMouseMove	release( )
enabled	name	onDragBegin	onOpen	setFocus( )
fontBold	pageno	onGotFocus	onRightDblClick	
fontItalic	parent	onHelp	onRightMouseDown	
fontName	printable	onLeftDblClick	onRightMouseUp	
fontSize	speedTip	onLeftMouseDown	when	
fontStrikeout	statusMessage	onLeftMouseUp		
fontUnderline	systemTheme	onLostFocus		
form	tabStop	onMiddleDblClick		
height	top			
helpFile	visible			
helpId	width			
hWnd				

Description

Use a CheckBox component to represent a *true*/*false* value.

class ColumnCheckBox

Topic group Related topics

A checkbox in a grid column.

Syntax

These controls are created by assigning the appropriate *editorType* to the GridColumn object.

Properties

The following tables list the properties of interest in the ColumnCheckBox class. (No particular events or methods are associated with this class.)

Property	Default	Description
baseClassName	COLUMNCHECKBOX	Identifies the object as an instance of the ColumnCheckBox class
className	(COLUMNCHECKBOX)	Identifies the object as an instance of custom class. When no custom class exists, defaults to baseClassName
colorHighlight		The color of the cell containing the ColumnCheckBox object when the cell has focus
colorNormal	WindowText /Window	The color of the cell containing the ColumnCheckBox object when the cell does not have focus
value		The current value of the check box (*true* or *false*)

The following table lists the common properties, events, and methods of the ColumnCheckBox class:

Property		Event		Method
hWnd	speedTip	beforeCellPaint	onMiddleDblClick	none
parent	statusMessage	onCellPaint	onMiddleMouseDown	
		onGotFocus	onMiddleMouseUp	
		onLeftDblClick	onMouseMove	

onLeftMouseDown
onLeftMouseUp
onLostFocus
onRightDblClick
onRightMouseDown
onRightMouseUp

Description

A ColumnCheckBox is a simplified CheckBox control in a grid column. When the enumerated *editorType* property of a GridColumn control is set to CheckBox, the column uses a ColumnCheckBox control, which is accessible through the GridColumn object's *editorControl* property.

By default, the checkbox around the checkmark is displayed for all grid cells in the column. This can be changed by toggling the grid property *alwaysDrawCheckBox* to false. When false, *alwaysDrawCheckBox* causes the grid to only draw the checkbox for the columnCheckBox cell that has focus. For columnCheckBox cells that do not have focus, there is only a checkmark if the value is true; or nothing if the value is false (the cell appears empty).

As with all column controls, the *dataLink* and *width* for the control is in the parent GridColumn object, not the control itself. The *height* is controlled by the *cellHeight* of the grid.

If a mouse event is implemented for this control it overrides the matching grid level event.

class ColumnComboBox

Topic group Related topics

A combobox in a grid column.

Syntax

These controls are created by assigning the appropriate *editorType* to the GridColumn object.

Properties

The following tables list the properties of interest in the ColumnComboBox class. (No particular events or methods are associated with this class.)

Property	Default	Description
autoTrim	false	whether or not trailing spaces are trimmed from character strings loaded from the control's dataSource.
baseClassName	COLUMNCOMBOBOX	Identifies the object as an instance of the ColumnComboBox class
className	(COLUMNCOMBOBOX)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorHighlight		The color of the text in the ColumnComboBox object when the object has focus
colorNormal	WindowText /Window	The color of the text in the ColumnComboBox object when the object does not have focus
dataSource		The option strings of the ColumnComboBox object
dropDownHeight		The number of options displayed in the drop-down list
dropDownWidth		The width of the drop-down list in the form's current metric units
function		A text formatting function

picture		Formatting template
sorted	false	Whether the options are sorted
value		The value currently displayed in the ColumnComboBox object

The following table lists the common properties, events, and methods of the ColumnComboBox class:

Property		Event		Method
borderStyle fontBold fontItalic fontName fontSize fontStrikeout fontUnderline	hWnd mousePointer parent speedTip statusMessage	onGotFocus onLeftDblClick onLeftMouseDown onLeftMouseUp onLostFocus	onMiddleDblClick onMiddleMouseDown onMiddleMouseUp onMouseMove onRightDblClick onRightMouseDown onRightMouseUp	none

Description

A ColumnComboBox is a simplified ComboBox control in a grid column. The combobox is always the DropDownList style. When the enumerated *editorType* property of a GridColumn control is set to ComboBox, the column uses a ColumnComboBox control, which is accessible through the GridColumn object's *editorControl* property.

Only the cell that has focus appears as a combobox. All other cells in the column which do not have focus appear as ColumnEntryfield controls instead, with no drop-down control.

As with all column controls, the *dataLink* and *width* for the control is in the parent GridColumn object, not the control itself. The *height* is controlled by the *cellHeight* of the grid.

If a mouse event is implemented for this control it overrides the matching grid level event.

class ColumnEditor

Topic group Related topics

An expandable editor object in a grid column used to enter or display data from memo, text blob or character fields.

Syntax

These controls are created by assigning the appropriate editorType to the GridColumn object.

Properties

The following tables list the properties and events of interest in the ColumnEditor class. (No methods are associated with this class.)

Property	Default	Description
baseClassName	COLUMNEDITOR	Identifies the object as an instance of the ColumnEditor class
className	(COLUMNEDITOR)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorHighlight		The color of the text in the ColumnEdItor object when the object has focus

colorNormal	WindowText /Window	The color of the text in the ColumnEditor object when the object does not have focus
dropDownHeight	8	The height of the ColumnEditor's dropdown editing window. The *dropDownHeight* property's value is in units matching the *metric* of the columnEditor's parent Form.
evalTags	true	Whether or not to apply embedded formatting tags when displaying the contents of the columnEditor's datalinked field.
value		The value currently displayed in the ColumnEditor object
wrap	true	Whether to word-wrap the text in the ColumnEditor control.

Event	Parameters	Description
key	<char expN>, <position expN>, <shift expL>, <ctrl expL>	When a key is pressed. Return value may change or cancel keystroke.
valid		When attempting to remove focus. Must return *true*, or focus remains.

The following table lists the common properties and events of the ColumnEditor class:

Property		Event	Method
fontBold	hWnd	beforeCellPaint	none
fontItalic	parent	onCellPaint	
fontName	speedTip	onGotFocus	
fontSize	statusMessage	onLostFocus	
fontStrikeout			
fontUnderline			

Description

A ColumnEditor object provides functionality similar to that of an Editor object, but in a grid cell. A ColumnEditor may be datalinked (via its parent gridColumn object) to a memo field, a text type blob field, or a character field.

When a ColumnEditor object has focus, a button is displayed which can be used to open an expanded, or drop-down window, in which to view or edit data. Clicking the mouse outside the expanded editor window, or pressing tab or shift-tab, will close the window.

When not expanded, ColumnEditor objects initially display the first non-blank line of data from its datalinked field. This is to make it easier for a user to determine what, if any, data has been entered into the field.

To enter or edit data in a ColumnEditor object:

1. Give it focus by clicking on it with a left mouse button, or by using the tab or arrow keys to move to it within the grid object.
    - Position the mouse where you wish to begin typing, and again click the left mouse button to display an insertion point. Alternatively you can just press any text key on the keyboard to begin entering text. Once the ColumnEditor has an insertion point it is said to be in "edit mode". When the ColumnEditor is in edit mode, the arrow keys will only work to scroll within the ColumnEditor.

To exit the ColumnEditor:

Click outside the ColumnEditor's cell

or

Navigate your way out using the tab, or shift-tab, keys.

Altering column dimensions

You may widen a ColumnEditor by widening its grid column.

You may change the height of a ColumnEditor's grid cell, to display more than one line of data in the grid cell, by changing the grid's *cellHeight* property.

You may change the height of a ColumnEditor's expanded window by changing its *dropDownHeight* value.

The expanded window will not expand beyond the edge of the ColumnEditor's Form or Subform.

Formatting text

When the ColumnEditor is contained within a Form whose *mdi* property is set to *true*, the Format Toolbar may be used to apply various formatting options to the text. To access the Format Toolbar:

2. Give the ColumnEditor focus
   - Right click on it to popup a context sensitive menu
   - Select "Show Format Toolbar"

By default, a ColumnEditor will apply any formatting embedded in its datalinked field, when displaying data. To turn this off:

Set the ColumnEditor's *evalTags* property to *false*

or

Uncheck "Apply Formatting" via it's right-click popup menu.

class ColumnEntryfield

Topic group Related topics

A single-line text input field in a grid column.

Syntax

These controls are created by assigning the appropriate *editorType* to the GridColumn object.

Properties

The following tables list the properties and events of interest in the ColumnEntryfield class. (No methods are associated with this class.)

Property	Default	Description
baseClassName	COLUMNENTRYFIELD	Identifies the object as an instance of the ColumnEntryfield class
className	(COLUMNENTRYFIELD)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorHighlight		The color of the text in the ColumnEntryfield object when the object has focus
colorNormal	WindowText /Window	The color of the text in the ColumnEntryfield object when the object does not have focus
function		A text formatting function
memoEditor		The memo editor control used when editing a memo field
picture		Formatting template
validErrorMsg	Invalid input	The message that is displayed when the *valid* event

	fails
value	The value currently displayed in the ColumnEntryfield object

Event	Parameters	Description
key	<char expN>, <position expN>, <shift expL>, <ctrl expL>	When a key is pressed. Return value may change or cancel keystroke.
valid		When attempting to remove focus. Must return *true*, or focus remains.

The following table lists the common properties, events, and methods of the ColumnEntryfield class:

Property		Event		Method
borderStyle fontBold fontItalic fontName fontSize fontStrikeout fontUnderline	hWnd parent speedTip statusMessage	beforeCellPaint onCellPaint onGotFocus onLeftDblClick onLeftMouseDown onLeftMouseUp onLostFocus	onMiddleDblClick onMiddleMouseDown onMiddleMouseUp onRightDblClick onRightMouseDown onRightMouseUp	none

Description

A ColumnEntryfield is a simplified Entryfield control in a grid column. When the enumerated *editorType* property of a GridColumn control is set to Entryfield, the column uses a ColumnEntryfield control, which is accessible through the GridColumn object's *editorControl* property.

As with all column controls, the *dataLink* and *width* for the control is in the parent GridColumn object, not the control itself. The *height* is controlled by the *cellHeight* of the grid.

If a mouse event is implemented for this control it overrides the matching grid level event.

class ColumnHeadingControl

Topic group Related topics

A grid column heading.

Syntax

These controls are created for each GridColumn object.

Properties

The following tables list the properties of interest in the ColumnHeadingControl class. (No particular events or methods are associated with this class.)

Property	Default	Description
baseClassName	COLUMNHEADINGCONTROL	Identifies the object as an instance of the ColumnHeadingControl class
className	(COLUMNHEADINGCONTROL)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to

		baseClassName
colorNormal	WindowText /Window	The color of the control and its text
function		A text formatting function
picture		Formatting template
value		The text displayed in the ColumnHeadingControl object

The following table lists the common properties, events, and methods of the ColumnHeadingControl class:

Property		Event		Method
fontBold	hWnd	beforeCellPaint	onMiddleMouseDown	none
fontItalic	parent	onCellPaint	onMiddleMouseUp	
fontName		onLeftDblClick	onRightDblClick	
fontSize		onLeftMouseDown	onRightMouseDown	
fontStrikeout		onLeftMouseUp	onRightMouseUp	
fontUnderline		onMiddleDblClick		

Description

Each column in a grid has a ColumnHeadingControl object that represents the column heading. It is accessible through the GridColumn object's *headingControl* property.

As with all column controls, the *width* for the control is in the parent GridColumn object, not the control itself. The *height* is controlled by the *cellHeight* of the grid.

If a mouse event is implemented for this control it overrides the matching grid level event.

class ColumnSpinBox

Topic group Related topics

An entryfield with a spinner for entering numeric or date values in a grid column.

Syntax

These controls are created by assigning the appropriate *editorType* to the GridColumn object.

Properties

The following tables list the properties and events of interest in the ColumnSpinBox class. (No methods are associated with this class.)

Property	Default	Description
baseClassName	COLUMNSPINBOX	Identifies the object as an instance of the ColumnSpinBox class
className	(COLUMNSPINBOX)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorHighlight		The color of the text in the ColumnSpinBox object when the object has focus
colorNormal	WindowText /Window	The color of the text in the ColumnSpinBox object when the object does not have focus
function		A text formatting function
picture		Formatting template

rangeMax		The maximum value
rangeMin		The minimum value
rangeRequired	false	Whether the range values are enforced even when no change has been made
step	1	The value added or subtracted when using the spinner
validErrorMsg	Invalid input	The message that is displayed when the *valid* event fails
value		The value currently displayed in the ColumnSpinBox object

Event	Parameters	Description
valid		When attempting to remove focus. Must return *true*, or focus remains.

The following table lists the common properties, events, and methods of the ColumnSpinBox class:

Property		Event		Method
borderStyle fontBold fontItalic fontName fontSize fontStrikeout fontUnderline	hWnd parent speedTip statusMessage	beforeCellPaint onCellPaint onGotFocus onLeftDblClick onLeftMouseDown onLeftMouseUp onLostFocus	onMiddleDblClick onMiddleMouseDown onMiddleMouseUp onMouseMove onRightDblClick onRightMouseDown onRightMouseUp	none

Description

A ColumnSpinBox is a simplified SpinBox control in a grid column. When the enumerated *editorType* property of a GridColumn control is set to SpinBox, the column uses a ColumnSpinBox control, which is accessible through the GridColumn object's *editorControl* property.

Only the cell that has focus appears as a spinbox. All other cells in the column which do not have focus appear as ColumnEntryfield controls instead, with no spinner control.

As with all column controls, the *dataLink* and *width* for the control is in the parent GridColumn object, not the control itself. The *height* is controlled by the *cellHeight* of the grid.

If a mouse event is implemented for this control it overrides the matching grid level event.

class ComboBox

Topic group Related topics

A component on a form which can be temporarily expanded to show a list from which you can pick a single item.

Syntax

[<oRef> =] new ComboBox(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created ComboBox object.

<container>

The container—typically a Form object—to which you're binding the ComboBox object.

<name expC>

An optional name for the ComboBox object. If not specified, the ComboBox class will auto-generate a name for the object.

Properties

The following tables list the properties, events, and methods of the ComboBox class.

Property	Default	Description
autoDrop	false	Whether the drop-down list automatically drops down when the combobox gets focus
baseClassName	COMBOBOX	Identifies the object as an instance of the ComboBox class
className	COMBOBOX	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorHighlight	WindowText /Window	The color of the text in the ComboBox object when it has focus
colorNormal	WindowText /Window	The color of the text in the ComboBox object when it does not have focus
dataLink		The Field object that is linked to the ComboBox object
dataSource		The option strings of the ComboBox object
dropDownHeight		The height of the drop-down list in the form's current metric units.
dropDownWidth		The width of the drop-down list in the form's current metric units
maxLength		Specifies the maximum number of characters allowed.
selectAll	true	Whether the selectAll behavior is used in the entry field portion of the ComboBox
sorted	false	Whether the options are sorted
style	DropDown	The style of the ComboBox: 0=Simple, 1=DropDown, 2=DropDownList
value		The value of the currently selected option

Event	Parameters	Description
beforeCloseUp		Fires just before dropdown list is closed for a style 1 or 2 combobox
beforeDropDown		Fires just before dropdown list opens for a style 1 or 2 combobox
beforeEditPaint		For a style 0 or 1 combobox, fires for each keystroke that modifies the value of the combobox, just before the new value is displayed
key	<char expN>, <position expN>, <shift expL>, <ctrl expL>	When a key is pressed. Return value may change or cancel keystroke.
onChange		Fires after the string in the ComboBox object has changed and the ComboBox object loses focus, but before *onLostFocus*
onChangeCancel		Fires when the user takes an action that closes the dropdown list without choosing an item from the list for a style 1 or 2 combobox
onChangeCommitted		Fires when the user takes an action to choose an item from the list such as by left clicking the mouse on an item or pressing Enter with an item highlighted.

onEditPaint		For a style 0 or 1 combobox, fires for each keystroke that modifies the value of the combobox, just after the new value is displayed
onKey	<char expN>, <position expN>, <shift expL>, <ctrl expL>	After a key has been pressed (and the *key* event has fired), but before the next keypress.

Method	Parameters	Description
copy( )		Copies selected text to the Windows clipboard
cut( )		Cuts selected text to the Windows clipboard
keyboard( )	<expC>	Simulates typed user input to the ComboBox object
paste( )		Copies text from the Windows clipboard to the current cursor position
undo( )		Reverses the effects of the most recent *cut*( ), *copy*( ), or *paste*( ) action

The following table lists the common properties, events, and methods of the ComboBox class:

Property		Event		Method
before	id	beforeRelease	onMiddleMouseDown	drag( )
borderStyle	left	onClose	onMiddleMouseUp	move( )
dragEffect	mousePointer	onDesignOpen	onMouseMove	release( )
enabled	name	onDragBegin	onOpen	setFocus( )
fontBold	pageno	onGotFocus	onRightDblClick	
fontItalic	parent	onHelp	onRightMouseDown	
fontName	printable	onLeftDblClick	onRightMouseUp	
fontSize	speedTip	onLeftMouseDown	when	
fontStrikeout	statusMessage	onLeftMouseUp		
fontUnderline	systemTheme	onLostFocus		
form	tabStop	onMiddleDblClick		
height	top			
helpFile	visible			
helpId	width			
hWnd				

Description

Use a ComboBox object when you want the user to pick one item from a list. When the user is not choosing an item, the list is not visible. The list of options is set with the *dataSource* property.

If a ComboBox is *dataLink*ed to a field object that has implemented its *lookupSQL* or *lookupRowset* properties, the ComboBox will automatically be populated with the appropriate lookup values, and store the corresponding key values in the *dataLink*ed field.

class Container

Topic group

A container for other controls.

Syntax

[<oRef> =] new Container(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created Container object.

<container>

The container—typically a Form object—to which you're binding the Container object.

<name expC>

An optional name for the Container object. If not specified, the Container class will auto-generate a name for the object.

Properties

The following table lists the properties of interest in the Container class. (No particular methods are associated with this class.)

Property	Default	Description
allowDrop	false	Whether dragged objects can be dropped in the Container
anchor	0 – None	How the Container object is anchored in its container (0=None, 1=Bottom, 2=Top, 3=Left, 4=Right, 5=Center, 6=Container)
baseClassName	CONTAINER	Identifies the object as an instance of the Container class
className	(CONTAINER)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorNormal	BtnFace	The background color
expandable	true	Reports only: whether the container expands to show all its components
transparent	false	Whether the container has the same background color or image as the its own container (usually the form)

Event	Parameters	Description
onDragEnter	<left expN> <top expN> <type expC> <name expC>	When the mouse enters the Container's display area during a Drag&Drop operation
onDragLeave		When the mouse leaves the Container's display area without having dropped an object
onDragOver	<left expN> <top expN> <type expC> <name expC>	While the mouse drags an object over the Container's display area during a Drag&Drop operation
onDrop	<left expN> <top expN> <type expC> <name expC>	When the mouse button is released over the Container's display area during a Drag&Drop operation

The following table lists the common properties, events, and methods of the Container class:

Property		Event		Method
before	mousePointer	beforeRelease	onMiddleMouseDown	drag( )
borderStyle	name	canRender	onMiddleMouseUp	move( )
dragEffect	pageno	onClose	onMouseMove	release( )
enabled	parent	onDesignOpen	onOpen	
first	printable	onDragBegin	onRender	
form	speedTip	onLeftDblClick	onRightDblClick	
height	systemTheme	onLeftMouseDown	onRightMouseDown	

hWnd
left
top
visible
width
onLeftMouseUp
onMiddleDblClick
onRightMouseUp

Description

Use the Container object to create groups of controls, a custom control that contains multiple controls, or to otherwise group controls in a form. When a control is dropped in a Container object, it becomes a child object of the Container object. Its *parent* property references the container, while its *form* property references the form.

To make the rectangle that contains the controls invisible, set the *borderStyle* property to None (3) and the *transparent* property to *true*.

When the Container's *enabled* property is set to "false", the enabled properties of all contained controls are likewise set to "false". When the Container's *enabled* property is set to "true", the enabled properties of the contained controls regain their individual settings.

class Editor

Topic group Related topics

A multiple-line text input field on a form.

Syntax

[<oRef> =] new Editor(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created Editor object.

<container>

The container—typically a Form object—to which you're binding the Editor object.

<name expC>

An optional name for the Editor object. If not specified, the Editor class will auto-generate a name for the object.

Properties

The following tables list the properties, events, and methods of interest in the Editor class.

Property	Default	Description
anchor	0 – None	How the Editor object is anchored in its container (0=None, 1=Bottom, 2=Top, 3=Left, 4=Right, 5=Center, 6=Container)
baseClassName	EDITOR	Identifies the object as an instance of the Editor class
border	true	Whether the Editor object is surrounded by the border specified by *borderStyle*
className	(EDITOR)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorHighlight	WindowText /Window	The color of text in the Editor object that is not affected by embedded color formatting while the Editor object has focus
colorNormal	WindowText /Window	The color of text in the Editor object that is not affected by embedded color formatting when the Editor object does not have focus
columnNo	1	The current column number in the Editor

cuaTab	true	Whether pressing Tab follows CUA behavior and moves to next control, or inserts tab in text
dataLink		The Field object that is linked to the Editor object
evalTags	true	Whether to evaluate any HTML formatting tags in the text or display them as-is
lineNo	1	The current line number in the editor
marginHorizontal		The horizontal margin between the text and its rectangular frame
marginVertical		The vertical margin between the text and its rectangular frame
modify	true	Whether the text is editable or not
popupEnable	true	Whether the Editor object's context menu is available
scrollBar	Auto	When a scroll bar appears for the Editor object (0=Off, 1=On, 2=Auto, 3=Disabled)
value		The string currently displayed in the Editor object
wrap	true	Whether to word-wrap the text in the editor

Event	Parameters	Description
key	<char expN>, <position expN>, <shift expL>, <ctrl expL>	When a key is pressed in the Editor. Return value may change or cancel keystroke.
onChange		After the string in the Editor object has changed and the Editor object loses focus, but before *onLostFocus*
onDrop	<left expN> <top expN> <type expC> <name expC>	When the mouse button is released over the Editor's display area during a Drag&Drop operation
valid		When attempting to remove focus. Must return *true*, or focus remains.

Method	Parameters	Description
copy( )		Copies selected text to the Windows clipboard
cut( )		Cuts selected text and to the Windows clipboard
keyboard( )	<expC>	Simulates typed user input to the Editor object
paste( )		Copies text from the Windows clipboard to the current cursor position
undo( )		Reverses the effects of the most recent cut( ), copy( ), or paste( ) action

The following table lists the common properties, events, and methods of the Editor class:

Property		Event		Method
before	id	beforeRelease	onMiddleMouseDown	drag( )
borderStyle	left	onClose	onMiddleMouseUp	move( )
dragEffect	mousePointer	onDesignOpen	onMouseMove	release( )
enabled	name	onDragBegin	onOpen	setFocus( )
fontBold	pageno	onGotFocus	onRightDblClick	
fontItalic	parent	onHelp	onRightMouseDown	
fontName	printable	onLeftDblClick	onRightMouseUp	
fontSize	speedTip	onLeftMouseDown	when	
fontStrikeout	statusMessage	onLeftMouseUp		
fontUnderline	systemTheme	onLostFocus		

form	tabStop	onMiddleDblClick
height	top	
helpFile	visible	
helpId	width	
hWnd		

Description

Use an Editor component to display and edit multi-line text. To display the text but not allow changes, set the *modify* property to *false*. The Editor component understands and displays basic HTML formatting tags. It has a context menu that is accessible by right-clicking the editor (unless its *popupEnable* property is *false*). The context menu lets you find and replace text, toggle word wrapping and HTML formatting, and show or hide the Format toolbar.

Supported HTML tags available in the editor:

Tag	Description
br	LineBreakTag
p	ParagraphTag
font	FontTag
i	ItalicTag
b	BoldTag
u	UnderlineTag
h1	Header1Tag
h2	Header2Tag
h3	Header3Tag
h4	Header4Tag
h5	Header5Tag
h6	Header6Tag
img	ImgTag
a	AnchorTag
strong	StrongTag
em	EmTag
big	BigTag
small	SmallTag
cite	CiteTag
address	AddressTag
strike	StrikeoutTag
tt	TypewriterTag
code	CodeTag
sub	SubscriptTag
sup	SuperscriptTag

blockquote	BlockQuoteTag
ul	UnorderedListTag
ol	OrderedListTag
li	ListItemTag
pre	PreformattedTag
font color	FontColorTag
font size	FontSizeTag
font face	FontFaceTag
a href	HyperLinkTag
a name	InternalAnchorTag
p align	ParaAlignTag
h1 align	Header1AlignTag
h2 align	Header2AlignTag
h3 align	Header3AlignTag
h4 align	Header4AlignTag
h5 align	Header5AlignTag
h6 align	Header6AlignTag
img src	ImgSrcTag
img width	ImgWidthTag
img height	ImgHeightTag
img align	ImgAlignTag
img border	ImgBorderTag
img alt	ImgAltTag
img hspace	ImgHorzSpace
img vspace	ImgVertSpace

class Entryfield

Topic group Related topics

A single-line text input field on a form.

Syntax

[<oRef> =] new Entryfield(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created Entryfield object.

<container>

The container—typically a Form object—to which you're binding the Entryfield object.

<name expC>

An optional name for the Entryfield object. If not specified, the Entryfield class will auto-generate a name for the object.

Properties

The following tables list the properties, events, and methods of interest in the Entryfield class.

Property	Default	Description
baseClassName	ENTRYFIELD	Identifies the object as an instance of the Entryfield class
border	true	Whether the Entryfield object is surrounded by the border specified by *borderStyle*
className	(ENTRYFIELD)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorHighlight		The color of the text in the Entryfield object when the object has focus
colorNormal	WindowText /Window	The color of the text in the Entryfield object when the object does not have focus
dataLink		The Field object that is linked to the Entryfield object
function		A text formatting function
maxLength		The maximum length of the text in the Entryfield object
memoEditor		The memo editor control used when editing a memo field
phoneticLink		The control that mirrors the phonetic equivalent of the current value
picture		Formatting template
selectAll	true	Whether the entryfield contents are initially selected when the Entryfield object gets focus
validErrorMsg	Invalid input	The message that is displayed when the *valid* event fails
validRequired	false	Whether to fire the *valid* event even when no change has been made
value		The value currently displayed in the Entryfield object

Event	Parameters	Description
key	<char expN>, <position expN>, <shift expL>, <ctrl expL>	When a key is pressed. Return value may change or cancel keystroke.
onChange		After the string in the Entryfield object has changed and the Entryfield object loses focus, but before *onLostFocus*
onKey	<char expN>, <position expN>, <shift expL>, <ctrl expL>	After a key has been pressed (and the *key* event has fired), but before the next keypress.
valid		When attempting to remove focus. Must return *true*, or focus remains.

Method	Parameters	Description
copy( )		Copies selected text to the Windows clipboard
cut( )		Cuts selected text and to the Windows clipboard
keyboard( )	<expC>	Simulates typed user input to the Entryfield object
paste( )		Copies text from the Windows clipboard to the current cursor position
showMemoEditor( )		Opens the specified *memoEditor*
undo( )		Reverses the effects of the most recent *cut*( ), *copy*( ), or *paste*( )

action

The following table lists the common properties, events, and methods of the Entryfield class:

Property		Event		Method
before	id	beforeRelease	onMiddleMouseDown	drag( )
borderStyle	left	onClose	onMiddleMouseUp	move( )
dragEffect	mousePointer	onDesignOpen	onMouseMove	release( )
enabled	name	onDragBegin	onOpen	setFocus( )
fontBold	pageno	onGotFocus	onRightDblClick	
fontItalic	parent	onHelp	onRightMouseDown	
fontName	printable	onLeftDblClick	onRightMouseUp	
fontSize	speedTip	onLeftMouseDown	when	
fontStrikeout	statusMessage	onLeftMouseUp		
fontUnderline	systemTheme	onLostFocus		
form	tabStop	onMiddleDblClick		
height	top			
helpFile	visible			
helpId	width			
hWnd				

Description

Entryfield objects are the primary data display and entry component.

class Form

Topic group Related topics

A Form object.

Syntax

[<oRef> =] new Form([<title expC>])

<oRef>

A variable or property in which to store a reference to the newly created Form object.

<title expC>

An optional title for the Form object. If not specified, the title will be "Form".

Properties

The following tables list the properties, events, and methods of the Form class. With the exception of the onClose event, all Form events require the form to be open in order to fire.

Property	Default	Description
activeControl		The currently active control
allowDrop	false	Whether dragged objects can be dropped on the Form's surface
appSpeedBar	2	Whether to hide or display the Standard Toolbar when a form recieves focus. 0=Hide, 1=Display, 2=Use the current _app object's speedBar setting.
autoCenter	false	Whether the form automatically centers on-screen when it is opened
autoSize	false	Whether the form automatically sizes itself to display all its components
background		Background image
baseClassName	FORM	Identifies the object as an instance of the Form class
className	(FORM)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName

clientEdge	false	Whether the edge of the form has the sunken client appearance
colorNormal	BtnFace	Background color
designView		A view that is used when designing the form
elements		An array containing object references to the components on the form
escExit	true	Whether pressing Esc closes the form
first		The first component on the form in the z-order
hWndClient		The Windows handle for the form's client area
hWndParent	0	When used in conjunction with the *showTaskBarButton* property; determines, or specifies, the *hWnd* property for the parent window of a form
icon		An icon file or resources that displays when the form is minimized
inDesign		Whether the form was instantiated by the Form designer
maximize	true	Whether the form can be maximized when not MDI
mdi	true	Whether the form is MDI or SDI
menuFile		The name of the form's .MNU menu file
metric	Chars	Units of measurement (0=Chars, 1=Twips, 2=Points, 3=Inches, 4=Centimeters, 5=Millimeters, 6=Pixels)
minimize	true	Whether the form can be minimized when not MDI
moveable	true	Whether the form is moveable when not MDI
nextObj		The object that's about to receive focus
persistent	false	Determines whether custom control, datamodule, menu or procedure files associated with a form are loaded in the persistent mode.
popupMenu		The form's Popup menu object
refreshAlways	true	Whether to refresh the form after all form-based navigation and updates
rowset		The primary rowset
scaleFontBold	false	Whether the base font used for the Chars *metric* is boldface
scaleFontName	Arial	The base font used for the Chars *metric*
scaleFontSize	10	The point size of the base font used for the Chars *metric*
scrollBar	Off	When a scroll bar appears for the form (0=Off, 1=On, 2=Auto, 3=Disabled)
scrollHOffset		The current position of the horizontal scrollbar in units matching the form or subform's current metric property
scrollVOffset		The current position of the vertical scrollbar in units matching the form or subform's current metric property
showSpeedTip	true	Whether to show tool tips
showTaskBarButton	true	Whether to display a button for the form on the Windows Taskbar
sizeable	true	Whether the form is resizeable when not MDI
smallTitle	false	Whether the form has the smaller palette-style title bar when not MDI
sysMenu	true	Whether the form's system menu icon and close icon are displayed when not MDI
text		The text that appears in the form's title bar
topMost	false	Whether the form stays on top when not MDI
useTablePopup	false	Whether to use the default table navigation popup when no popup is assigned as the form's *popupMenu*.

view		The query or table on which the form is based
windowState	Normal	The state of the window (0=Normal, 1=Minimized, 2=Maximized)

Event	Parameters	Description
canClose		When attempting to close form; return value allows or disallows closure
canNavigate	<workarea expN>	When attempting to navigate in work area; return value allows or disallows leaving current record
onAppend		After a new record is added
onChange	<workarea expN>	After leaving a record that was changed, before *onNavigate*
onClose		After the form has been closed
onDragEnter	<left expN> <top expN> <type expC> <name expC>	When the mouse enters the Form's display area during a Drag&Drop operation
onDragLeave		When the mouse leaves the Form's display area without having dropped an object
onDragOver	<left expN> <top expN> <type expC> <name expC>	While the mouse drags an object over the Form's display area during a Drag&Drop operation
onDrop	<left expN> <top expN> <type expC> <name expC>	When the mouse button is released over the Form's display area during a Drag&Drop operation
onMove		After the form has been moved
onNavigate	<workarea expN>	After navigation in a work area
onSelection	<control ID expN>	After the form is submitted
onSize	<expN>	After the form is resized or changes *windowState*

Method	Parameters	Description
abandonRecord( )		Abandons changes to the current record
beginAppend( )		Starts append of new record
close( )		Closes the form
isRecordChanged( )		Checks whether the current record buffer has changed
open( )		Loads and opens the form
pageCount( )		Returns the highest pageno of any component
print( )		Prints the form
readModal( )		Opens the form modally
refresh( )		Redraws the form
saveRecord( )		Saves changes to the current or new record
scroll( )	<horizontal expN>, <vertical expN>	Programatically scrolls the client area (the contents) of a form
showFormatBar( )	<expL>	Displays or hides the formatting toolbar

The following table lists the common properties, events, and methods of the Form class:

Property		Event		Method

enabled
height
helpFile
helpId
hWnd
left
mousePointer
pageno
statusMessage
systemTheme
top
visible
width
beforeRelease
onClose
onDesignOpen
onGotFocus
onHelp
onLeftDblClick
onLeftMouseDown
onLeftMouseUp
onLostFocus
onMiddleDblClick
onMiddleMouseDown
onMiddleMouseUp
onMouseMove
onMouseOut
onMouseOver
onOpen
onRightDblClick
onRightMouseDown
onRightMouseUp
move( )
release( )
setFocus( )

Description

A Form object acts as a container for other visual components (also known as controls) and the data objects that are linked to them. Consequently, releasing a form object from memory automatically releases the objects it contains.

An object reference to all the visual components in a form is stored in its *elements* array. All of the visual components have a *form* property that points back to the form.

The form has a *rowset* property that refers to its primary rowset. Components can access this rowset in their event handlers generically with the object reference *form.rowset*. For example, a button on a form that goes to the first row in the rowset would have an *onClick* event handler like this:

```
function firstButton_onClick()
form.rowset.first()
```

Note: With the exception of the onClose event, all Form events require the form to be open in order to fire.

If the form has more than one rowset, each one can be addressed through the *rowset* property of the Query objects, which are properties of the form. For example, to go to the last row in the rowset of the Query object members1, the *onClick* event handler would look like this:

```
function lastMemberButton_onClick()
form.members1.rowset.last()
```

A form can consist of more than one page. One way to implement multi-page forms is to use the *pageno* property of controls to determine on which page they appear, and use a TabBox control to let users easily switch between pages. You may also use a NoteBook control to create a multi-page container in a form.

You can create two types of forms: modal and modeless. A modal form halts execution of the routine that opened it until the form is closed. When active, it takes control of the user interface; users can't switch to another window in the same application without exiting the form. A dialog box is an example of a modal form; when it is opened, program execution stops and focus can't be given to another window until the user closes the dialog box.

In contrast a modeless form window allows users to freely switch to other windows in an application. Most forms that you create for an application will be modeless. A modeless form window conforms to the Multiple Document Interface (MDI) protocol, which lets you open multiple document windows within an application window.

To create and use a modeless form, set the *mdi* property to *true* and open the form with the *open*( ) method. To create and use a modal form, set *mdi* to *false* and open the form with the *readModal*( ) method.

You can also create SDI (Single Document Interface) windows that appear like application windows. To do so, set the *mdi* property to *false* and use SHELL(*false*). SHELL(*false*) hides the standard *dBASE Plus* environment and lets your form take over the user interface. The *dBASE Plus* application window disappears, and the form name appears in the Windows Task List.

class Grid

Topic group Related topics

A grid of other controls.

Syntax

[<oRef> =] new Grid(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created Grid object.

<container>

The container—typically a Form object—to which you're binding the Grid object.

<name expC>

An optional name for the Grid object. If not specified, the Grid class will auto-generate a name for the object.

Properties

The following tables list the properties, events, and methods of interest in the Grid class.

Property	Default	Description
allowAddRows	true	Whether navigating down past the last row automatically calls *beginAppend*( )
allowColumnMoving	true	Whether columns may be moved with the mouse
allowColumnSizing	true	Whether columns may be sized with the mouse
allowDrop	false	Whether dragged objects (normally a table or table field) can be dropped in the Grid
allowEditing	true	Whether editing is allowed or the grid is read-only
allowRowSizing	true	Whether rows may be sized with the mouse
alwaysDrawCheckBox	true	Whether columnCheckBox control is painted with a checkbox for all checkbox cells in the Grid.
anchor	0 – None	How the Grid object is anchored in its container (0=None, 1=Bottom, 2=Top, 3=Left, 4=Right, 5=Center, 6=Container)
baseClassName	GRID	Identifies the object as an instance of the Grid class
bgColor	gray	Sets the background color for data displayed in grid cells, as well as the empty area to the right of the last column and below the last grid row.
cellHeight		The height of each cell
className	(GRID)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorColumnLines	silver	Sets the color of the grid lines between the data columns
colorHighlight	WindowText/Window	Sets the text color and background color for data displayed in a grid cell that has focus. Can be overridden by setting the colorHighlight property of a GridColumn's editorControl to a non-null value.
colorNormal	WindowText/Window	Sets the text color and background color for data displayed in grid cells that do not have focus. Can be overridden by setting the colorNormal property of a

		GridColumn's editorControl to a non-null value.
colorRowHeader	WindowText/BtnFace	Sets the color of the indicator arrow, or plus sign, and the row header background.
colorRowLines	siilver	Sets the color of the grid lines between the data rows
colorRowSelect	HighlightText/HighLight	Sets the text color and background color for a row of data selected when the rowSelect property and/or the multiSelect property is true
columnCount		The number of columns in the grid
columns		An array of objects for each column in the grid
cuaTab	false	Whether pressing Tab follow CUA behavior and moves to next control, or moves to next cell
currentColumn		The number of the column that has focus in the grid
dataLink		The Rowset object that is linked to the grid
dragScrollRate	300	The delay time, in milliseconds, between each column scroll when dragging columns
firstColumn	1	Sets the column to be displayed in the left-most unlocked column position.
frozenColumn		The name of the column inside which the cursor is confined.
gridLineWidth	1	Width of grid lines in pixels (0=no grid lines)
hasColumnHeadings	true	Whether column headings are displayed
hasColumnLines	true	Whether column (vertical) grid lines are displayed
hasIndicator	true	Whether the indicator column is displayed
hasRowLines	true	Whether row (horizontal) grid lines are displayed
hasVScrollHintText	true	Whether the relative row count is displayed as the grid is scrolled vertically
headingColorNormal	WindowText/BtnFace	Sets the text color and background color for grid column heading controls
headingFontBold	true	Whether the current heading font style is Bold
headingFontItalic	false	Whether the current heading font style is Italic
headingFontName	Operating system or PLUS.ini file setting	Sets the font used to display data in a grid's headingControls
headingFontSize	10 pts.	Sets the character size of the font used to display data in a grid's headingControls
headingFontStrikeout	false	Whether to display the current heading font with a horizontal strikeout line through the middle of each character
headingFontUnderline	false	Whether the current heading font style is Underline
hScrollBar	Auto	When a horizontal scrollbar appears (0=Off, 1=On, 2=Auto, 3=Disabled)
integralHeight	false	Whether a partial row at the bottom of the grid is displayed
lockedColumns	0	The number of columns that remain locked on the left side of the grid as it is scrolled horizontally.
multiSelect	false	Whether multiple rows may be visually selected
rowSelect	false	Whether the entire row is visually selected
vScrollBar	Auto	When a vertical scrollbar appears (0=Off, 1=On, 2=Auto, 3=Disabled)

Event	Parameters	Description
onDragEnter	<left expN> <top expN> <type expC> <name expC>	When the mouse enters the Grid display area during a Drag&Drop operation
onDragLeave		When the mouse leaves the Grid display area without having dropped an object
onDragOver	<left expN> <top expN> <type expC> <name expC>	While the mouse drags an object over the Grid display area during a Drag&Drop operation
onDrop	<left expN> <top expN> <type expC> <name expC>	When the mouse button is released over the Grid display area during a Drag&Drop operation
onFormSize		After the form containing the grid is resized
onSelChange		After moving to another row or column in the grid

Method	Parameters	Description
firstRow( )		Returns a bookmark for the row currently displayed in the first row of the grid.
getColumnObject( )	<expN>	Returns a reference to the GridColumn object for a designated column
getColumnOrder( )		Returns a two dimensional array for current column information
lastRow( )		Returns a bookmark for the row currently displayed in the last row of the grid.
refresh( )		Repaints the grid
selected( )		Returns an array of bookmarks for the currently selected rows in the grid

The following table lists the common properties, events, and methods of the Grid class:

Property		Event		Method
before	hWnd	beforeRelease	onMiddleMouseDown	drag( )
borderStyle	id	onClose	onMiddleMouseUp	move( )
dragEffect	left	onDesignOpen	onMouseMove	release( )
enabled	name	onDragBegin	onOpen	setFocus( )
fontBold	pageno	onGotFocus	onRightDblClick	
fontItalic	parent	onHelp	onRightMouseDown	
fontName	printable	onLeftDblClick	onRightMouseUp	
fontSize	speedTip	onLeftMouseDown		
fontStrikeout	systemTheme	onLeftMouseUp		
fontUnderline	tabStop	onLostFocus		
form	top	onMiddleDblClick		
height	visible			
helpFile	width			
helpId				

Description

The Grid object is a multi-column grid control for displaying the contents of a rowset. The *dataLink* property is set to the rowset. Columns are automatically created for each field in the rowset.

Each column is represented by a GridColumn object. If the default columns are used, these objects are hidden, and all fields are displayed. By explicitly creating a GridColumn object for each column as an element in the grid's *columns* array, you may control the fields that are displayed and assign different kinds of controls in different columns.

Navigation in the rowset updates any grids that are *dataLink*ed to the rowset, and vice versa. When you explicitly create GridColumn objects, you may set their *dataLink* properties to fields in other rowsets, like the fields in a linked detail table.

Grid level mouse event handlers will fire anywhere ona grid as long as the event handler is defined and is not overridden by a matching columnHeading or editorControl event. this includes mouse events on a column header, row header, grid cell, or grid background.

If you wish to have the Grid level events fire along with the editorControl or columnHeading control event, you can call the grid level event from within the editorControl or columnHeading control event handler.

While in Desgin Mode, if columns are defined, you can:

- size columns, move columns, and set the grid's cellHeight (rowHeight) by using the mouse
- select a column's editorControl or headingControl into the inspector by left clicking them with the mouse.

class GridColumn

Topic group Related topics

A column in a grid.

Syntax

[<oRef> =] new GridColumn(<grid>)

<oRef>

A variable or property—typically an array element of the <grid> object's *columns* array—in which to store a reference to the newly created GridColumn object.

<grid>

The Grid object that contains the GridColumn object.

Properties

The following tables list the properties of interest of the GridColumn class. (No particular events or methods are associated with this class.)

Property	Default	Description
baseClassName	GRIDCOLUMN	Identifies the object as an instance of the GridColumn class
className	(GRIDCOLUMN)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
dataLink		The field object that is linked to the column in the grid
editorControl		The editable control that comprises the body of the grid in the column

editorType	Default	The type of editing control (0=Default, 1=EntryField, 2=CheckBox, 3=SpinBox, 4=ComboBox, 5=Editor) in the column
headingControl		The control that displays the grid column heading

The following table lists the common properties, events, and methods of the GridColumn class:

Property		Event		Method
parent	width	none		none

Description

Each column in a grid is represented by a GridColumn object. Each GridColumn object is an element in the grid's *columns* array, and contains a reference to a heading control and an edit control. You may assign different kinds of controls in different columns. The following types of controls are supported:

Entryfield
CheckBox
SpinBox
ComboBox
Editor

When these controls are used in a grid, they have a reduced property set. Each type of field has a default control type. Logical and boolean fields default to CheckBox. Numeric and date fields default to SpinBox.

class Image

Topic group Related topics

A rectangular region on a form that displays a bitmap image.

Syntax

[<oRef> =] new Image(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created Image object.

<container>

The container—typically a Form object—to which you're binding the Image object.

<name expC>

An optional name for the Image object. If not specified, the Image class will auto-generate a name for the object.

Properties

The following table lists the properties of interest in the Image class. (No particular methods are associated with this class.)

Property	Default	Description
alignment	Stretch	Determines the size and position of the graphic inside the Image object (0=Stretch, 1=Top left, 2=Centered, 3=Keep aspect stretch, 4=True size)
allowDrop	false	Whether dragged objects (i.e. the name of a graphic image file) can be dropped in the Image object

anchor	0 – None	How the Image object is anchored in its container (0=None, 1=Bottom, 2=Top, 3=Left, 4=Right, 5=Center, 6=Container)
baseClassName	IMAGE	Identifies the object as an instance of the Image class
className	(IMAGE)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
dataSource		The file or field that is displayed in the Image object
fixed	false	Whether the Image object's position is fixed or if it can be "pushed down" or "pulled up" by the rendering or suppression of other objects
imgPixelHeight	0	Returns an image's actual height in pixels
imgPixelWidth	0	Returns an image's actual width in pixels

Event	Parameters	Description
onDragEnter	<left expN> <top expN> <type expC> <name expC>	When the mouse enters the Image object's display area during a Drag&Drop operation
onDragLeave		When the mouse leaves the Image object's display area without having dropped an object
onDragOver	<left expN> <top expN> <type expC> <name expC>	While the mouse drags an object over the Image object's display area during a Drag&Drop operation
onDrop	<left expN> <top expN> <type expC> <name expC>	When the mouse button is released over the Image object's display area during a Drag&Drop operation

The following table lists the common properties, events, and methods of the Image class:

Property		Event		Method
before	name	beforeRelease	onMiddleMouseDown	drag( )
borderstyle	pageno	canRender	onMiddleMouseUp	move( )
dragEffect	parent	onClose	onMouseMove	release( )
enabled	printable	onDesignOpen	onOpen	
form	speedTip	onDragBegin	onRender	
height	systemTheme	onLeftDblClick	onRightDblClick	
hWnd	top	onLeftMouseDown	onRightMouseDown	
id	visible	onLeftMouseUp	onRightMouseUp	
left	width	onMiddleDblClick		
mousePointer				

Description

Use an Image object to display a bitmap image. The image can be data from a field, or a static image like a company logo.

dBASE Plus supports the following bitmap image formats:

- Graphics Interchange Format (GIF), including animated GIF
- Joint Photographic Experts Group (JPG, JPEG)
- Portable Network Graphics (PNG)
- X BitMap (XBM)
- Windows bitmap (BMP)
- Windows icon (ICO)
- Device Independent Bitmap (DIB)

Windows metafile (WMF)
Enhanced Windows metafile (EMF)
PC Paintbrush (PCX)
Tag Image File Format (TIF, TIFF)
Encapsulated PostScript (EPS)

dBASE Plus will resize images according to the Image object's *alignment* property. When resizing, transparent GIF backgrounds are lost. To prevent resizing, set the *alignment* property to 4 (True size).

For TIFF, *dBASE Plus* supports uncompressed, single-bit Group 3, PackBits, and LZW (Lempel-Ziv & Welch) compression. Group 4 compression is not supported. Color TIFF images must have a palette. Except when rendering an EPS file on a PostScript-capable printer, *dBASE Plus* uses the bitmap preview in the EPS file, which must be in TIFF or WMF format.

class Line

Topic group Related topics

A line on a form.

Syntax

[<oRef> =] new Line(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created Line object.

<container>

The container—typically a Form object—to which you're binding the Line object.

<name expC>

An optional name for the Line object. If not specified, the Line class will auto-generate a name for the object.

Properties

The following tables list the properties of interest of the Line class. (No particular events or methods are associated with this class.)

Property	Default	Description
baseClassName	LINE	Identifies the object as an instance of the Line class
bottom		The location of the bottom end of the Line in the form's current metric units, relative to the top edge of its container
className	(LINE)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorNormal	WindowText	Color of the line
fixed	false	Whether the Line object's position is fixed or if it can be "pushed down" or "pulled up" by the rendering or suppression of other objects
left		The location of the left end of the Line in the form's current metric units, relative to the left edge of its container
pen	Solid	The pen style used to draw the line (0=Solid, 1=Dash, 2=Dot, 3=DashDot, 4=DashDotDot)
right		The location of the right end of the Line in the form's current metric units, relative to the left edge of its container

top		The location of the top of the Line in the form's current metric units, relative to the top edge of its container
width	1	Width in pixels

The following table lists the common properties, events, and methods of the Line class:

Property		Event	Method
before	pageno	beforeRelease	release( )
form	parent	canRender	
id	printable	onClose	
name	visible	onDesignOpen	
		onOpen	
		onRender	

Description

Use a Line object to draw a line in a form or report. Note that the position properties—*top*, *left*, *bottom*, and *right*—work different for the Line object than they do with other components. The *width* property controls the thickness of the line.

A Line has no *hWnd* because it is drawn on the surface of the form; it is not a genuine Windows control. Despite its position in the form's z-order, a Line can never be drawn on top of another component (other than a Line or Shape).

class ListBox

Topic group Related topics

A selection list on a form, from which you can pick multiple items.

Syntax

[<oRef> =] new ListBox(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created ListBox object.

<container>

The container—typically a Form object—to which you're binding the ListBox object.

<name expC>

An optional name for the ListBox object. If not specified, the ListBox class will auto-generate a name for the object.

Properties

The following tables list the properties, events, and methods of interest in the ListBox class.

Property	Default	Description
allowDrop	false	Whether dragged objects can be dropped in the ListBox object
baseClassName	LISTBOX	Identifies the object as an instance of the ListBox class
className	(LISTBOX)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorHighlight	HighlightText /Highlight	The color of selected options
colorNormal	WindowText /Window	The color of unselected options

curSel		The number of the option that has the focus rectangle
dataSource		The options strings of the ListBox object
multiple	false	Whether the ListBox object allows selection of more than one option
sorted	false	Whether the options are sorted
transparent	false	Whether the ListBox object has the same background color or image as its container
value		The value of the option that currently has focus
vScrollBar	Auto	When a vertical scrollbar appears (0=Off, 1=On, 2=Auto, 3=Disabled)

Event	Parameters	Description
key	<char expN>, <position expN>, <shift expL>, <ctrl expL>	When a key is pressed in the ListBox. Return value may change or cancel keystroke.
onDragEnter	<left expN> <top expN> <type expC> <name expC>	When the mouse enters the ListBox display area during a Drag&Drop operation
onDragLeave		When the mouse leaves the ListBox display area without having dropped an object
onDragOver	<left expN> <top expN> <type expC> <name expC>	While the mouse drags an object over the ListBox display area during a Drag&Drop operation
onDrop	<left expN> <top expN> <type expC> <name expC>	When the mouse button is released over the ListBox display area during a Drag&Drop operation
onKey	<char expN>, <position expN>, <shift expL>, <ctrl expL>	After a key has been pressed (and the *key* event has fired), but before the next keypress.
onSelChange		After the focus moves to another option in the list

Method	Parameters	Description
count( )		Returns the number of options in the list
selected( )		Returns the currently selected option(s) or checks if a specified option is selected

The following table lists the common properties, events, and methods of the ListBox class:

Property		Event		Method
before	hWnd	beforeRelease	onMiddleMouseDown	drag( )
borderStyle	id	onClose	onMiddleMouseUp	move( )
dragEffect	left	onDesignOpen	onMouseMove	release( )
enabled	mousePointer	onDragBegin	onOpen	setFocus( )
fontBold	name	onGotFocus	onRightDblClick	
fontItalic	pageno	onHelp	onRightMouseDown	
fontName	parent	onLeftDblClick	onRightMouseUp	
fontSize	printable	onLeftMouseDown	when	
fontStrikeout	speedTip	onLeftMouseUp		
fontUnderline	statusMessage	onLostFocus		

form
height
helpFile
helpId
systemTheme
tabStop
top
visible
width
onMiddleDblClick

Description

Use a ListBox object to present the user with a scrollable list of items. If the *multiple* property is *true*, the user can choose more than one item. The list of options is set with the *dataSource* property. The list of items selected is returned by calling the *selected*( ) method.

class NoteBook

Topic group

A multi-page container with rectangular tabs on top.

Syntax

[<oRef> =] new NoteBook(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created NoteBook object.

<container>

The container—typically a Form object—to which you're binding the NoteBook object.

<name expC>

An optional name for the NoteBook object. If not specified, the NoteBook class will auto-generate a name for the object.

Properties

The following tables list the properties and events of interest in the NoteBook class. (No particular methods are associated with this class.)

Property	Default	Description
allowDrop	false	Whether dragged objects can be dropped on the NoteBook
anchor	0 – None	How the NoteBook object is anchored in its container (0=None, 1=Bottom, 2=Top, 3=Left, 4=Right, 5=Center, 6=Container)
baseClassName	NOTEBOOK	Identifies the object as an instance of the NoteBook class
buttons	false	Whether the notebook tabs appear as buttons instead
className	(NOTEBOOK)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorNormal	BtnFace	Color of the notebook background
curSel		The number of the currently selected tab
dataSource		The tab names for the notebook
focus	Normal	When to give focus to the notebook tabs when they are clicked (0=Normal, 1=On Button Down, 2=Never)
multiple	false	Whether the notebook tabs are displayed in multiple rows, or in a single row with a scrollbar
visualStyle	Right Justify	The style of the notebook tabs (0=Right Justify, 1=Fixed Width, 2=Ragged Right)

Event	Parameters	Description
canSelChange	<nNewSel expN>	Before a different notebook tab is selected; return value determines if selection can leave the current tab.
onDragEnter	<left expN> <top expN> <type expC> <name expC>	When the mouse enters the notebook display area during a Drag&Drop operation
onDragLeave		When the mouse leaves the notebook display area without having dropped an object
onDragOver	<left expN> <top expN> <type expC> <name expC>	While the mouse drags an object over the notebook display area during a Drag&Drop operation
onDrop	<left expN> <top expN> <type expC> <name expC>	When the mouse button is released over the notebook display area during a Drag&Drop operation
onSelChange		After a different notebook tab is selected

The following table lists the common properties, events, and methods of the NoteBook class:

Property		Event		Method
before	id	beforeRelease	onMiddleMouseDown	drag( )
borderStyle	left	onClose	onMiddleMouseUp	move( )
dragEffect	mousePointer	onDesignOpen	onMouseMove	release( )
enabled	name	onDragBegin	onOpen	setFocus( )
first	pageno	onGotFocus	onRightDblClick	
fontBold	parent	onHelp	onRightMouseDown	
fontItalic	printable	onLeftDblClick	onRightMouseUp	
fontName	speedTip	onLeftMouseDown		
fontSize	statusMessage	onLeftMouseUp		
fontStrikeout	systemTheme	onLostFocus		
fontUnderline	tabStop	onMiddleDblClick		
form	top			
height	visible			
helpFile	width			
helpId				
hWnd				

Description

The NoteBook object combines aspects of the Form, Container, and TabBox objects. It's a multi-page control, like the Form; it acts as a container, and it has tabs, although they're on top. Selecting a tab automatically changes the page of the notebook to display the controls assigned to that page. The notebook's *pageno* property indicates which page of the form the notebook is in. The notebook's *curSel* property indicates the current page the notebook is displaying.

When the Notebook's *enabled* property is set to "false", the enabled properties of all contained controls are likewise set to "false". When the Notebook's *enabled* property is set to "true", the enabled properties of the contained controls regain their individual settings.

class OLE

Topic group Related topics

Displays an OLE document that is stored in an OLE field, and lets the user initiate an action in the server application that created the document.

Syntax

[<oRef> =] new OLE(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created OLE object.

<container>

The container—typically a Form object—to which you're binding the OLE object.

<name expC>

An optional name for the OLE object. If not specified, the OLE class will auto-generate a name for the object.

Properties

The following tables list the properties, events, and methods of interest in the OLE class.

Property	Default	Description
alignment	Stretch	Determines the size and position of the contents of the OLE object (0=Stretch, 1=Top left, 2=Centered, 3=Keep aspect stretch, 4=True size)
anchor	0 – None	How the OLE object is anchored in its container (0=None, 1=Bottom, 2=Top, 3=Left, 4=Right, 5=Center, 6=Container)
baseClassName	OLE	Identifies the object as an instance of the OLE class
border	false	Whether the OLE object is surrounded by the border specified by *borderStyle*
className	(OLE)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
dataLink		The Field object that is linked to the OLE object
linkFileName		The OLE document file (if any) that is linked with the current OLE field.
oleType		Number that reflects whether an OLE field is empty (0), contains an embedded document (1), or contains a link to a document file (2)
serverName		The server application that is invoked when the user double-clicks on an OLE viewer object

Event	Parameters	Description
onChange		After the contents of the OLE object have changed
onClose		After the form containing the OLE object has been closed

Method	Parameters	Description
doVerb( )	<OLE verb expN>, <title expC>	Starts an OLE server session

The following table lists the common properties, events, and methods of the OLE class:

Property		Event	Method
before	name	onDesignOpen	drag( )
borderStyle	pageno	onDragBegin	release( )
dragEffect	parent	onGotFocus	setFocus( )
enabled	printable	onLostFocus	
form	speedTip	onOpen	
height	statusMessage		

hWnd
id
left
mousePointer
systemTheme
tabStop
top
visible
width

Description

Place an OLE object in a form to view and edit a document stored in an OLE field. For example, if an OLE field contains a bitmap image created in Paintbrush, double-clicking the OLE object linked to the field starts a session in Paintbrush and places the image in the Paintbrush work area.

OLE stands for object linking and embedding. When you link a document to an OLE object, the OLE field does not contain the document itself; instead, it holds a link to a file containing the document. When you embed a document in an OLE field, a copy of the document is inserted into the OLE field, and no connection is made to a document file.

By double-clicking the OLE object, the user can invoke the application that created the OLE document. Therefore, if an image was created in Paintbrush and linked or embedded in the OLE field, double-clicking on the field starts a session in Paintbrush; the image is displayed in the Paintbrush drawing area, ready for editing. If the object was linked, any changes made in the Paintbrush session are stored in the document file; if the object was embedded, the changes are stored in the OLE field only.

An OLE viewer window object displays the contents of an OLE field. (Use the *dataLink* property to identify the field.) Each time the record pointer is moved, the contents of the viewer window are refreshed to display the OLE field in the current record.

class PaintBox

Topic group

A generic control that can be placed on a form.

Syntax

[<oRef> =] new PaintBox(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created PaintBox object.

<container>

The container—typically a Form object—to which you're binding the PaintBox object.

<name expC>

An optional name for the PaintBox object. If not specified, the PaintBox class will auto-generate a name for the object.

Properties

The following tables list the properties and events of interest in the PaintBox class. (No particular methods are associated with this class.)

Property	Default	Description
allowDrop	false	Whether dragged objects can be dropped in the PaintBox
baseClassName	PAINTBOX	Identifies the object as an instance of the PaintBox class

className	(PAINTBOX)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorNormal	BtnText/BtnFace	The color of the paintbox
transparent	false	Whether the paintbox background is the same as the background color or image of its container

Event	Parameters	Description
onChar	<char expN>, <repeat expN>, <flags expN>	After a non-cursor key or key combination is pressed
onClose		After the form containing the PaintBox object has been closed
onDragEnter	<left expN> <top expN> <type expC> <name expC>	When the mouse enters the paintbox display area during a Drag&Drop operation
onDragLeave		When the mouse leaves the paintbox display area without having dropped an object
onDragOver	<left expN> <top expN> <type expC> <name expC>	While the mouse drags an object over the paintbox display area during a Drag&Drop operation
onDrop	<left expN> <top expN> <type expC> <name expC>	When the mouse button is released over the paintbox display area during a Drag&Drop operation
onFormSize		After the form containing the paintbox is resized
onKeyDown	<char expN>, <repeat expN>, <flags expN>	After any key is pressed
onKeyUp	<char expN>, <repeat expN>, <flags expN>	After any key is released
onPaint		Whenever the paintbox needs to be redrawn

The following table lists the common properties, events, and methods of the PaintBox class:

Property		Event		Method
before	name	beforeRelease	onMiddleMouseDown	drag( )
borderStyle	pageno	onClose	onMiddleMouseUp	move( )
dragEffect	parent	onDesignOpen	onMouseMove	release( )
enabled	printable	onDragBegin	onMouseOut	setFocus( )
form	speedTip	onGotFocus	onMouseOver	
height	statusMessage	onLeftDblClick	onOpen	
hWnd	systemTheme	onLeftMouseDown	onRightDblClick	
id	tabStop	onLeftMouseUp	onRightMouseDown	
left	top	onLostFocus	onRightMouseUp	
mousePointer	visible	onMiddleDblClick		
	width			

Description

The PaintBox object is a generic control you can use to create a variety of objects. It is designed for advanced developers who want to create their own custom controls using the Windows API. It is simply a rectangular region of a form that has all the standard control properties such as *height*, *width*, and *before*, as well as all the standard mouse events.

In addition to the standard events or properties, the PaintBox object has three events that let you detect keystrokes entered when it has focus: *onChar*, *onKeyDown*, and *onKeyUp*. These let you create customized editing controls. The *onPaint* and *onFormSize* events let you modify the appearance of the object based on user interaction.

class Progress

Topic group

A progress indicator.

Syntax

[<oRef> =] new Progress(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created Progress object.

<container>

The container—typically a Form object—to which you're binding the Progress object.

<name expC>

An optional name for the Progress object. If not specified, the Progress class will auto-generate a name for the object.

Properties

The following tables list the properties of interest in the Progress class. (No particular events or methods are associated with this class.)

Property	Default	Description
baseClassName	PROGRESS	Identifies the object as an instance of the Progress class
className	(PROGRESS)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
rangeMax		The maximum value
rangeMin		The minimum value
value		The current value

The following table lists the common properties, events, and methods of the Progress class:

Property		Event		Method
before	pageno	beforeRelease	onMiddleMouseDown	drag( )
borderStyle	parent	onClose	onMiddleMouseUp	move( )
dragEffect	printable	onDesignOpen	onMouseMove	release( )
form	speedTip	onDragBegin	onOpen	
height	systemTheme	onLeftMouseDown	onRightMouseDown	
hWnd	top	onLeftMouseUp	onRightMouseUp	
left	visible			
mousePointer	width			
name				

Description

Use a Progress object to graphically indicate progress during processing. For example to display percentage completed, set the *rangeMin* to 0 and the *rangeMax* to 100. Then as the

process progresses, set the *value* to the approximate percentage. The control will display the percentage graphically.

class PushButton

Topic group

A button on a form.

Syntax

[<oRef> =] new PushButton(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created PushButton object.

<container>

The container—typically a Form object—to which you're binding the PushButton object.

<name expC>

An optional name for the PushButton object. If not specified, the PushButton class will auto-generate a name for the object.

Properties

The following tables list the properties and events of interest in the PushButton class. (No particular methods are associated with this class.)

Property	Default	Description
baseClassName	PUSHBUTTON	Identifies the object as an instance of the PushButton class
bitmapAlignment	0 (Default)	Controls position of bitmaps and text on the pushButton
className	(PUSHBUTTON)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorNormal	BtnText/BtnFace	The color of the button
default	false	Whether the button is the default button on the form
disabledBitmap		The bitmap to display on the button when it's disabled
downBitmap		The bitmap to display on the button when it's pressed down
focusBitmap		The bitmap to display on the button when it has focus
group		The group to which the button belongs
speedBar	false	Whether the button acts like a tool button, which never gets focus
systemTheme	true	Whether to use XP Visual Style button, or Windows Classic button
text	<same as name>	The text that appears on the PushButton face
textLeft	false	When a button has both a bitmap and text label, whether the text appears to the left or right of the bitmap
toggle	false	Whether the button acts like a toggle switch, staying down when pressed
upBitmap	0	The bitmap to display on the button when it's not down and does not have focus
value	false	Whether the button is pressed (used when *toggle* is *true*)

Event	Parameters	Description
onClick		After the PushButton is clicked

The following table lists the common properties, events, and methods of the PushButton class:

Property		Event		Method
before	id	beforeRelease	onMiddleMouseDown	drag( )
borderStyle	left	onClose	onMiddleMouseUp	move( )
dragEffect	mousePointer	onDesignOpen	onMouseMove	release( )
enabled	name	onDragBegin	onOpen	setFocus( )
fontBold	pageno	onGotFocus	onRightDblClick	
fontItalic	parent	onHelp	onRightMouseDown	
fontName	printable	onLeftDblClick	onRightMouseUp	
fontSize	speedTip	onLeftMouseDown	when	
fontStrikeout	statusMessage	onLeftMouseUp		
fontUnderline	systemTheme	onLostFocus		
form	tabStop	onMiddleDblClick		
height	top			
helpFile	visible			
helpId	width			
hWnd				

Description

Use a PushButton object to execute an action when the user clicks it.

class RadioButton

Topic group Related topics

A single RadioButton on a form. The user may choose one from a set.

Syntax

[<oRef> =] new RadioButton(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created RadioButton object.

<container>

The container—typically a Form object—to which you're binding the RadioButton object.

<name expC>

An optional name for the RadioButton object. If not specified, the RadioButton class will auto-generate a name for the object.

Properties

The following tables list the properties and events of interest in the RadioButton class. (No particular methods are associated with this class.)

Property	Default	Description
baseClassName	RADIOBUTTON	Identifies the object as an instance of the RadioButton class
className	(RADIOBUTTON)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorNormal	BtnText/BtnFace	The color of the RadioButton's text label
dataLink		The Field object that is linked to the RadioButton

group		The group to which the RadioButton belongs
text	<same as name>	The text label that appears beside the RadioButton
textLeft	false	Whether the RadioButton's text label appears to the left or to the right of the RadioButton
transparent	false	Whether the RadioButton object has the same background color or image as its container
value		Whether the RadioButton is visually marked as selected

Event	Parameters	Description
onChange		After the RadioButton gets selected or loses its selection

The following table lists the common properties, events, and methods of the RadioButton class:

Property		Event		Method
before	id	beforeRelease	onMiddleMouseDown	drag( )
borderStyle	left	onClose	onMiddleMouseUp	move( )
dragEffect	mousePointer	onDesignOpen	onMouseMove	release( )
enabled	name	onDragBegin	onOpen	setFocus( )
fontBold	pageno	onGotFocus	onRightDblClick	
fontItalic	parent	onHelp	onRightMouseDown	
fontName	printable	onLeftDblClick	onRightMouseUp	
fontSize	speedTip	onLeftMouseDown	when	
fontStrikeout	statusMessage	onLeftMouseUp		
fontUnderline	systemTheme	onLostFocus		
form	tabStop	onMiddleDblClick		
height	top			
helpFile	visible			
helpId	width			
hWnd				

Description

Use a group of RadioButton objects to present the user a set of multiple choices, from which they can choose only one.

Each set of choices on a form must have the same *group* property. If there is only one group of RadioButtons on a form, the *group* can be left blank. You may use any string or number as the *group* property.

You may also use *true* and *false* in the *group* property to create RadioButton groups. Use *true* for the first button in each RadioButton group, and *false* for the rest. For example, if you create seven RadioButtons and set the *group* property of the first and fourth RadioButton to *true*, the first three buttons form one group, and the last four form another. The two groups are independent; the user can select one button in the first group and one button in the other.

class Rectangle

Topic group Related topics

A rectangle with a caption.

Syntax

[<oRef> =] new Rectangle(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created Rectangle object.

<container>

The container—typically a Form object—to which you're binding the Rectangle object.

<name expC>

An optional name for the Rectangle object. If not specified, the Rectangle class will auto-generate a name for the object.

Properties

The following tables list the properties of interest of the Rectangle class. (No particular events or methods are associated with this class.)

Property	Default	Description
baseClassName	RECTANGLE	Identifies the object as an instance of the Rectangle class
border	true	Whether the Rectangle object's rectangle is visible
borderStyle	Default	Specifies the rectangle style (0=Default, 1=Raised, 2=Lowered, 3=None, 4=Single, 5=Double, 6=Drop Shadow, 7=Client, 8=Modal, 9=Etched In, 10=Etched Out)
className	(RECTANGLE)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorNormal	BtnText/BtnFace	The color of the caption and the rectangle fill
patternStyle	Solid	The fill pattern style (0=Solid, 1=BDiagonal, 2=Cross, 3=Diagcross, 4=FDiagonal, 5=Horizontal, 6=Vertical)
text	<same as name>	The text caption that appears at the top right of the rectangle
transparent	false	Determines if the interior of rectangle is, or is not, transparent.

The following table lists the common properties, events, and methods of the Rectangle class:

Property		Event		Method
before	hWnd	beforeRelease	onMiddleMouseDown	drag( )
borderStyle	left	onClose	onMiddleMouseUp	move( )
dragEffect	mousePointer	onDesignOpen	onMouseMove	release( )
fontBold	name	onDragBegin	onOpen	
fontItalic	pageno	onLeftDblClick	onRightDblClick	
fontName	parent	onLeftMouseDown	onRightMouseDown	
fontSize	printable	onLeftMouseUp	onRightMouseUp	
fontStrikeout	speedTip	onMiddleDblClick		
fontUnderline	systemTheme			
form	top			
height	visible			
	width			

Description

Use a Rectangle object to enclose an area of a form. For example, you can use a Rectangle object to draw a border around a group of related objects, such as a group of RadioButtons.

To assign a label that describes the group of objects, use the *text* property. The label appears in the top left corner of the rectangle.

A Rectangle object does not affect other objects. The user can't give focus to the Rectangle object, and it doesn't display or modify data.

class ReportViewer

Topic group

A control to display a report on a form.

Syntax

[<oRef> =] new ReportViewer(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created ReportViewer object.

<container>

The container—typically a Form object—to which you're binding the ReportViewer object.

<name expC>

An optional name for the ReportViewer object. If not specified, the ReportViewer class will auto-generate a name for the object.

Properties

The following tables list the properties, events and methods of interest in the ReportViewer class.

Property	Default	Description
allowDrop	false	Whether dragged objects (normally an .REP file) can be dropped in the ReportViewer
anchor	0 – None	How the ReportViewer object is anchored in its container (0=None, 1=Bottom, 2=Top, 3=Left, 4=Right, 5=Center, 6=Container)
baseClassName	REPORTVIEWER	Identifies the object as an instance of the ReportViewer class
className	(REPORTVIEWER)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
filename		The name of the .REP file containing the report to view
params		Parameters passed to the .REP file
ref		A reference to the Report object being viewed
scrollBar	Auto	When a scroll bar appears for the ReportViewer object (0=Off, 1=On, 2=Auto, 3=Disabled)

Event	Parameters	Description
onDragEnter	<left expN> <top expN> <type expC> <name expC>	When the mouse enters the ReportViewer display area during a Drag&Drop operation
onDragOver	<left expN> <top expN> <type expC> <name expC>	While the mouse drags an object over the ReportViewer display area during a Drag&Drop operation
onDragLeave		When the mouse leaves the ReportViewer display area without having dropped an object
onDrop	<left expN> <top expN> <type expC> <name expC>	When the mouse button is released over the ReportViewer display area during a Drag&Drop operation

onLastPage		When the last page of a report has been rendered in the reportViewer

Method	Parameters	Description
reExecute( )		Regenerates the report

The following table lists the common properties, events, and methods of the ReportViewer class:

Property		Event	Method
before	name	beforeRelease	drag( )
borderStyle	pageno	onClose	move( )
dragEffect	parent	onDesignOpen	release( )
form	systemTheme	onDragBegin	
height	top	onOpen	
left	width		

Description

Use a ReportViewer object to view a report in a form. Assign any parameters to the *params* property, then set the *filename* property to the name of the .REP file; this executes the named report file. You may access the report object being viewed through the *ref* property.

If report parameters are assigned after setting the *filename* property, you must call the *reExecute*( ) method to regenerate the report.

class ScrollBar

Topic group Related topics

A vertical or horizontal scrollbar used to represent a range of numeric values.

Syntax

[<oRef> =] new ScrollBar(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created ScrollBar object.

<container>

The container—typically a Form object—to which you're binding the ScrollBar object.

<name expC>

An optional name for the ScrollBar object. If not specified, the ScrollBar class will auto-generate a name for the object.

Properties

The following tables list the properties and events of interest in the ScrollBar class. (No particular methods are associated with this class.)

Property	Default	Description
baseClassName	SCROLLBAR	Identifies the object as an instance of the ScrollBar class
className	(SCROLLBAR)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorNormal	ScrollBar	The color of the scrollbar

dataLink		The Field object that is linked to the ScrollBar object
rangeMax		The maximum value
rangeMin		The minimum value
value		The current value
vertical	true	Whether the scrollbar is vertical or horizontal

Event	Parameters	Description
onChange		After the scrollbar value changes

The following table lists the common properties, events, and methods of the ScrollBar class:

Property		Event		Method
before	name	beforeRelease	onMiddleMouseDown	drag( )
borderStyle	pageno	onClose	onMiddleMouseUp	move( )
dragEffect	parent	onDesignOpen	onMouseMove	release( )
enabled	printable	onDragBegin	onOpen	setFocus( )
form	speedTip	onGotFocus	onRightDblClick	
height	statusMessage	onHelp	onRightMouseDown	
helpFile	systemTheme	onLeftDblClick	onRightMouseUp	
helpId	tabStop	onLeftMouseDown	when	
hWnd	top	onLeftMouseUp		
id	visible	onLostFocus		
left	width	onMiddleDblClick		
mousePointer				

Description

The ScrollBar object is maintained primarily for compatibility. Use a Slider instead.

class Shape

Topic group Related topics

A simple colored geometric shape.

Syntax

[<oRef> =] new Shape(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created Shape object.

<container>

The container—typically a Form object—to which you're binding the Shape object.

<name expC>

An optional name for the Shape object. If not specified, the Shape class will auto-generate a name for the object.

Properties

The following tables list the properties of interest of the Shape class. (No particular events or methods are associated with this class.)

Property	Default	Description
baseClassName	SHAPE	Identifies the object as an instance of the Shape class

className	(SHAPE)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorNormal	WindowText /Window	The pen and fill color of the shape
drawMode	0 - Normal	An enumerated property used to create visual effects using the pen and fill color of the shape, and the color of the underlying object. See drawMode for details.
penStyle	Solid	The pen style used to draw the outline of the shape (0=Solid, 1=Dash, 2=Dot, 3=DashDot, 4=DashDotDot)
penWidth	1	Width of the outline in pixels
shapeStyle	Circle	The type of shape to draw (0=Round Rectangle, 1=Rectangle, 2=Ellipse, 3=Circle, 4=Round square, 5=Square)

The following table lists the common properties, events, and methods of the Shape class:

Property		Event	Method
before	pageno	beforeRelease	drag( )
dragEffect	parent	canRender	move( )
form	printable	onClose	release( )
height	top	onDesignOpen	
left	visible	onDragBegin	
name	width	onOpen	
		onRender	

Description

Use a Shape object to draw a basic geometric shape on a form.

A Shape has no *hWnd* because it is drawn on the surface of the form; it is not a genuine Windows control. Despite its position in the form's z-order, a Shape can never be drawn on top of another component (other than a Line or Shape).

class Slider

Topic group

A horizontal or vertical slider for choosing magnitude.

Syntax

[<oRef> =] new Slider(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created Slider object.

<container>

The container—typically a Form object—to which you're binding the Slider object.

<name expC>

An optional name for the Slider object. If not specified, the Slider class will auto-generate a name for the object.

Properties

The following tables list the properties, events, and methods of interest in the Slider class.

Property	Default	Description

baseClassName	SLIDER	Identifies the object as an instance of the Slider class
className	(SLIDER)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorNormal	BtnFace	The color of the slider
enableSelection	false	Whether to display the selection range
endSelection		The value of the end of the selection range
rangeMax		The maximum value
rangeMin		The minimum value
startSelection		The value of the start of the selection range
tics	Auto	How to display the tic marks (0=Auto, 1=Manual, 2=None)
ticsPos	Bottom Right	Where to display the tic marks (0=Both, 1=Bottom Right, 2=Top Left)
value		The current value
vertical	false	Whether the slider is vertical or horizontal

Event	Parameters	Description
onChange		After the slider position changes

Method	Parameters	Description
clearTics( )	<expN>	If <expN> is non-zero, clears all manually-set tic marks
setTic( )	<expN>	Manually sets a tic mark at the specified position
setTicFrequency( )	<expN>	Sets the automatic tic mark interval

The following table lists the common properties, events, and methods of the Slider class:

Property		Event		Method
before	name	beforeRelease	onMiddleMouseDown	drag( )
borderStyle	pageno	onClose	onMiddleMouseUp	move( )
dragEffect	parent	onDesignOpen	onMouseMove	release( )
enabled	printable	onDragBegin	onOpen	setFocus( )
form	speedTip	onGotFocus	onRightMouseDown	
height	statusMessage	onHelp	onRightMouseUp	
helpFile	systemTheme	onLeftMouseDown	when	
helpId	tabStop	onLeftMouseUp		
hWnd	top	onLostFocus		
id	visible			
left	width			
mousePointer				

Description

Use a slider to let users vary numeric values visually. Unlike spinboxes, sliders don't accept keyboard input or use a step value. Instead, the user drags the slider pointer to increase or decrease the value.

As the user moves the slider pointer, the value is continually updated to reflect the position of the pointer. For example, a slider that varies a numeric value between 1 and 100 sets the value to 50 when the slider pointer is at the center of the slider.

To set a range for the slider, set *rangeMin* to the minimum value and *rangeMax* to the maximum value.

You may also designate a separate selection region inside the slider with the *startSelection*, *endSelection*, and *enableSelection* properties.

You have complete control over the tick marks that appear in the slider.

class SpinBox

Topic group Related topics

An entryfield with a spinner for entering numeric or date values.

Syntax

[<oRef> =] new SpinBox(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created SpinBox object.

<container>

The container—typically a Form object—to which you're binding the SpinBox object.

<name expC>

An optional name for the SpinBox object. If not specified, the SpinBox class will auto-generate a name for the object.

Properties

The following tables list the properties, events, and methods of interest in the SpinBox class.

Property	Default	Description
baseClassName	SPINBOX	Identifies the object as an instance of the SpinBox class
border	true	Whether the SpinBox object is surrounded by the border specified by *borderStyle*
className	SPINBOX	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorHighlight		The color of the text in the SpinBox object when the object has focus
colorNormal	WindowText /Window	The color of the text in the SpinBox object when the object does not have focus
dataLink		The Field object that is linked to the SpinBox object
function		A text formatting function
picture		Formatting template
rangeMax		The maximum value
rangeMin		The minimum value
rangeRequired	false	Whether the range values are enforced even when no change has been made
selectAll	true	Whether the entryfield contents are initially selected when the SpinBox object gets focus
spinOnly	false	Whether the value may changed using the spinner only or typing is allowed
step	1	The value added or subtracted when using the spinner
validErrorMsg	Invalid input	The message that is displayed when the *valid* event fails

validRequired	false	Whether to fire the *valid* event even when no change has been made
value		The value currently displayed in the SpinBox object

Event	Parameters	Description
key	<char expN>, <position expN>, <shift expL>, <ctrl expL>	When a key is pressed in the entryfield portion of the spinbox. Return value may change or cancel keystroke.
onChange		After the spinner is clicked
onKey	<char expN>, <position expN>, <shift expL>, <ctrl expL>	After a key has been pressed (and the *key* event has fired), but before the next keypress.
valid		When attempting to remove focus. Must return *true*, or focus remains.

Method	Parameters	Description
copy( )		Copies selected text to the Windows Clipboard
cut( )		Cuts selected text to the Windows Clipboard
keyboard( )	<expC>	Simulates typed user input to the SpinBox object
paste( )		Copies text from the Windows Clipboard to the current cursor position
undo( )		Reverses the effects of the most recent *cut*( ), *copy*( ), or *paste*( ) action

The following table lists the common properties, events, and methods of the SpinBox class:

Property		Event		Method
before	id	beforeRelease	onMiddleMouseDown	drag( )
borderStyle	left	onClose	onMiddleMouseUp	move( )
dragEffect	mousePointer	onDesignOpen	onMouseMove	release( )
enabled	name	onDragBegin	onOpen	setFocus( )
fontBold	pageno	onGotFocus	onRightDblClick	
fontItalic	parent	onHelp	onRightMouseDown	
fontName	printable	onLeftDblClick	onRightMouseUp	
fontSize	speedTip	onLeftMouseDown	when	
fontStrikeout	statusMessage	onLeftMouseUp		
fontUnderline	systemTheme	onLostFocus		
form	tabStop	onMiddleDblClick		
height	top			
helpFile	visible			
helpId	width			
hWnd				

Description

Use a spinbox to let users enter values by typing them in the textbox or by clicking the spinner arrows.

By setting *spinOnly* to *true*, you can control the rate at which users change numeric or date values. For example, one spin box might change an interest rate in increments of hundredths, while another might change a date value in week increments. Set the size of each increment with the *step* property; for example, if you set *step* to 5, each click on an arrow changes a numeric value by 5 or a date value by 5 days.

To restrict entries to those within a particular range of values, set the *rangeMin* property to the minimum value and *rangeMax* to the maximum value, then set *rangeRequired* to *true*.

class SubForm

Topic group Related topics

A subclassed Form which behaves as a non-mdi form. A subform can be a child of a form or another subform object.

Syntax

[<oRef> =] new SubForm(<parent oRef>[<title expC>])

<oRef>

A variable or property in which to store a reference to the newly created SubForm object.

<parent oRef>

A variable or property containing an object reference to the form, or subform, that is to be the parent of the new subform. Determines the read-only value of the subform's *parent* property.

<title expC>

An optional title for the SubForm object. If not specified, the title will be "SubForm".

Properties

The following tables list the properties, events, and methods of the SubForm class.

Property	Default	Description
activeControl		The currently active control
allowDrop	false	Whether dragged objects can be dropped on the subform's surface
autoCenter	false	Whether the form automatically centers on-screen when it is opened
autoSize	false	Whether the form automatically sizes itself to display all its components
background		Background image
baseClassName	SUBFORM	Identifies the object as an instance of the SubForm class
className	(SUBFORM)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
clientEdge	false	Whether the edge of the form has the sunken client appearance
colorNormal	BtnFace	Background color
elements		An array containing object references to the components on the subform
escExit	true	Whether pressing Esc closes the subform
first		The first component on the subform in the z-order
hWndClient		The Windows handle for the subform's client area
icon		An icon file or resources that displays when the subform is minimized
inDesign		Whether the subform was instantiated by the Form designer
maximize	true	Whether the subform can be maximized when not MDI
metric	Chars	Units of measurement (0=Chars, 1=Twips, 2=Points, 3=Inches, 4=Centimeters, 5=Millimeters, 6=Pixels)
minimize	true	Whether the subform can be minimized when not MDI

moveable	true	Whether the subform is moveable when not MDI
nextObj		The object that's about to receive focus
persistent	false	Determines whether custom control, datamodule, menu or procedure files associated with a subform are loaded in the persistent mode.
popupMenu		The subform's Popup menu object
refreshAlways	true	Whether to refresh the subform after all form-based navigation and updates
rowset		The primary rowset
scrollBar	Off	When a scroll bar appears for the subform (0=Off, 1=On, 2=Auto, 3=Disabled)
scrollHOffset		The current position of the horizontal scrollbar in units matching the form or subform's current metric property
scrollVOffset		The current position of the vertical scrollbar in units matching the form or subform's current metric property
showSpeedTip	true	Whether to show tool tips
sizeable	true	Whether the subform is resizeable when not MDI
smallTitle	false	Whether the subform has the smaller palette-style title bar when not MDI
sysMenu	true	Whether the subform's system menu icon and close icon are displayed when not MDI
text		The text that appears in the subform's title bar
topMost	false	Whether the subform stays on top when not MDI
useTablePopup	false	Whether to use the default table navigation popup when no popup is assigned as the subform's *popupMenu*.
view		The query or table on which the subform is based
windowState	Normal	The state of the window (0=Normal, 1=Minimized, 2=Maximized)

Event	Parameters	Description
canClose		When attempting to close subform; return value allows or disallows closure
canNavigate	<workarea expN>	When attempting to navigate in work area; return value allows or disallows leaving current record
onAppend		After a new record is added
onChange	<workarea expN>	After leaving a record that was changed, before *onNavigate*
onClose		After the subform has been closed
onDragEnter	<left expN> <top expN> <type expC> <name expC>	When the mouse enters the subform's display area during a Drag&Drop operation
onDragLeave		When the mouse leaves the subform's display area without having dropped an object
onDragOver	<left expN> <top expN> <type expC> <name expC>	While the mouse drags an object over the subform's display area during a Drag&Drop operation
onDrop	<left expN> <top expN> <type expC> <name expC>	When the mouse button is released over the subform's display area during a Drag&Drop operation

onMove		After the subform has been moved
onNavigate	<workarea expN>	After navigation in a work area
onSelection	<control ID expN>	After the subform is submitted
onSize	<expN>	After the subform is resized or changes *windowState*

Method	Parameters	Description
abandonRecord( )		Abandons changes to the current record
beginAppend( )		Starts append of new record
close( )		Closes the subform
isRecordChanged( )		Checks whether the current record buffer has changed
open( )		Loads and opens the subform
pageCount( )		Returns the highest pageNo of any component
print( )		Prints the subform
refresh( )		Redraws the subform
saveRecord( )		Saves changes to the current or new record
scroll( )	<horizontal expN>, <vertical expN>	Programatically scrolls the client area (the contents) of a subform
showFormatBar( )	<expL>	Displays or hides the formatting toolbar

The following table lists the common properties, events, and methods of the SubForm class:

Property		Event		Method
enabled	pageNo	beforeRelease	onMiddleMouseDown	onRightMouseUp
height	parent	onClose	onMiddleMouseUp	move( )
helpFile	statusMessage	onDesignOpen	onMouseMove	release( )
helpId	systemTheme	onGotFocus	onMouseOut	setFocus( )
hWnd	top	onHelp	onMouseOver	
left	visible	onLeftDblClick	onOpen	
mousePointer	width	onLeftMouseDown	onRightDblClick	
		onLeftMouseUp	onRightMouseDown	
		onLostFocus		
		onMiddleDblClick		

Description

Parenting the subform to a form, or another subform, restricts display of the subform to within the client area of the parent form. When a parent form is closed, it allows the parent form to also close the subform.

A form or subform, containing one or more subforms, internally tracks which subform (if any) is currently active. When a subform is active, that subform has focus. When a form object is given focus, the active subform will lose focus and be set to inactive. Clicking on a subform, or subform object, will activate the subform and set focus either to the subform or the selected object.

A form's *canClose* event will call the *canClose* event of any child subforms. If a subform's *canClose* event returns *false*, the form's *canClose* event will also return *false*.

Subforms are not currently supported by the Form Designer. However, you can design a form in the Form Designer and edit the streamed code to designate it a Subform.

class TabBox

Topic group Related topics

A set of folder-style (trapezoidal) bottom-tabs.

Syntax

[<oRef> =] new TabBox(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created TabBox object.

<container>

The container—typically a Form object—to which you're binding the TabBox object.

<name expC>

An optional name for the TabBox object. If not specified, the TabBox class will auto-generate a name for the object.

Properties

The following tables list the properties and events of interest in the TabBox class. (No particular methods are associated with this class.)

Property	Default	Description
anchor	1 – Bottom	How the TabBox object is anchored in its container (0=None, 1=Bottom)
baseClassName	TABBOX	Identifies the object as an instance of the TabBox class
className	(TABBOX)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorHighlight	BtnText/BtnFace	The color of the selected tab
colorNormal	BtnFace	The color of the background behind the tabs
curSel		The number of the currently selected tab
dataSource		The tab names

Event	Parameters	Description
onSelChange		After a different tab is selected

The following table lists the common properties, events, and methods of the TabBox class:

Property		Event		Method
before	id	beforeRelease	onMiddleMouseDown	drag( )
dragEffect	left	onClose	onMiddleMouseUp	move( )
enabled	mousePointer	onDesignOpen	onMouseMove	release( )
fontBold	name	onDragBegin	onOpen	setFocus( )
fontItalic	pageno	onGotFocus	onRightDblClick	
fontName	parent	onHelp	onRightMouseDown	
fontSize	printable	onLeftDblClick	onRightMouseUp	
fontStrikeout	speedTip	onLeftMouseDown	when	
fontUnderline	statusMessage	onLeftMouseUp		
form	systemTheme	onLostFocus		
height	tabStop	onMiddleDblClick		
helpFile	top			
helpId	visible			
hWnd	width			

Description

A TabBox contains a number of tabs that users can select.

By setting the *pageno* property of a TabBox control to zero (the default), you can implement a tabbed multi-page form where the user can easily switch pages by selecting tabs. Use the *pageno* property of a control to determine on which page the control appears, and use the *curSel* property and *onSelChange* event of the TabBox to switch between pages.

class Text

Topic group Related topics

Non-editable HTML text on a form.

Syntax

[<oRef> =] new Text(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created Text object.

<container>

The container—typically a Form object—to which you're binding the Text object.

<name expC>

An optional name for the Text object. If not specified, the Text class will auto-generate a name for the object.

Properties

The following tables list the properties and methods of interest in the Text class. (No particular events are associated with this class.)

Property	Default	Description
alignHorizontal	Left	Determines how the text displays within the horizontal plane of its rectangular frame (0=Left, 1=Center, 2=Right, 3=Justify)
alignment	Top left	Combines the *alignHorizontal*, *alignVertical*, and *wrap* properties (maintained for compatibility)
alignVertical	Top	Determines how the text displays in the vertical plane of its rectangular frame (0=Top, 1=Center, 2=Bottom, 3=Justify)
anchor	0 – None	How the Text object is anchored in its container (0=None, 1=Bottom, 2=Top, 3=Left, 4=Right, 5=Center, 6=Container)
baseClassName	TEXT	Identifies the object as an instance of the Text class
border	false	Whether the Text object is surrounded by the border specified by *borderStyle*
className	TEXT	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorNormal	BtnText/BtnFace	The color of the text
fixed	false	Whether the Text object's position is fixed or if it can be "pushed down" or "pulled up" by the rendering or suppression of other objects

function		A text formatting function
leading	0	The distance between consecutive lines; if 0 uses the font's default leading
marginHorizontal		The horizontal margin between the text and its rectangular frame
marginVertical		The vertical margin between the text and its rectangular frame
picture		Formatting template
prefixEnable	true	Whether to interpret the ampersand (&) character in the *text* as the accelerator prefix.
rotate	0	The text orientation, in increments of 90 degrees (0=0, 1=90, 2=180, 3=270)
suppressIfBlank	false	Whether the Text object is suppressed (not rendered) if it is blank
suppressIfDuplicate	false	Whether the Text object is suppressed (not rendered) if its value is the same as the previous time it was rendered
text	<same as name>	The value of the Text object; the text that appears
tracking	0	The space between characters; if zero uses the font's default
trackJustifyThreshold	0	The maximum amount of added space between words on a fully justified line; zero indicates no limit
transparent	false	Whether the Text object has the same background color or image as its container
variableHeight	false	Whether the Text object's height can increase based on its value
verticalJustifyLimit	0	The maximum additional space between lines that can be added to attempt to justify vertically. If the limit is exceeded the Text object is top justified. A value of zero means no limit.
wrap	true	Whether to word-wrap the text in the Text object

Method	Parameters	Description
getTextExtent( )	<expC>	Returns the width of the specified string using the Text object's font

The following table lists the common properties, events, and methods of the Text class:

Property		Event		Method
before	id	beforeRelease	onMiddleMouseDown	drag( )
borderStyle	left	canRender	onMiddleMouseUp	move( )
dragEffect	mousePointer	onClose	onMouseMove	release( )
fontBold	name	onDesignOpen	onOpen	
fontItalic	pageno	onDragBegin	onRender	
fontName	parent	onLeftDblClick	onRightDblClick	
fontSize	printable	onLeftMouseDown	onRightMouseDown	
fontStrikeout	speedTip	onLeftMouseUp	onRightMouseUp	
fontUnderline	systemTheme	onMiddleDblClick		
form	top			
height	visible			
hWnd	width			

Description

Use a Text component to display information in a form or report. The *text* property of the component may contain any text, including HTML tags. Use a TextLabel component in forms where the extended functionality of the Text component is not required.

The *text* property may be an expression codeblock, which is evaluated every time it is rendered.

Note

The properties, *marginHorizontal*, *marginVertical*, *suppressfBlank*, *suppressfDuplicate*, *tracking*, *trackJustifyThreshold*, *verticalJustifyLimit* and *variableHeight* are designed to be used only in reports.

class TextLabel

Topic group Related topics

Non-editable text on a form.

Syntax

[<oRef> =] new TextLabel(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created TextLabel object.

<container>

The container—typically a Form object—to which you're binding the TextLabel object.

<name expC>

An optional name for the TextLabel object. If not specified, the TextLabel class will auto-generate a name for the object.

Properties

The following tables list the properties and methods of interest in the TextLabel class. (No particular events are associated with this class.)

Property	Default	Description
alignHorizontal	Left	Determines how the text displays within the horizontal plane of its rectangular frame (0=Left, 1=Center, 2=Right)
alignVertical	Top	Determines how the text displays in the vertical plane of its rectangular frame (0=Top, 1=Center, 2=Bottom)
baseClassName	TEXTLABEL	Identifies the object as an instance of the TextLabel class
className	(TEXTLABEL)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorNormal	BtnText/BtnFace	The color of the text
prefixEnable	true	Whether to interpret the ampersand (&) character in the *text* as the accelerator prefix.
text	<same as *name*>	The value of the TextLabel object; the text that appears
transparent	false	Whether the TextLabel object has the same background color or image as its container

Method	Parameters	Description
getTextExtent( )	<expC>	Returns the width of the specified string using the TextLabel object's font

The following table lists the common properties, events, and methods of the TextLabel class:

Property		Event		Method
before	id	beforeRelease	onMiddleMouseDown	drag( )
borderStyle	left	onClose	onMiddleMouseUp	move( )
dragEffect	mousePointer	onDesignOpen	onMouseMove	release( )
fontBold	name	onDragBegin	onOpen	
fontItalic	pageno	onLeftDblClick	onRightDblClick	
fontName	parent	onLeftMouseDown	onRightMouseDown	
fontSize	printable	onLeftMouseUp	onRightMouseUp	
fontStrikeout	speedTip	onMiddleDblClick		
fontUnderline	systemTheme			
form	top			
height	visible			
hWnd	width			

Description

Use a TextLabel component to display information on a form or report, wherever features such as word-wrap and HTML formatting are not required. TextLabel is a simple, light-duty object which consumes fewer system resources than the Text component.

The TextLabel component does not support in-place editing on design surfaces.

The *text* property of the TextLabel component may contain character string data only.

class TreeItem

Topic group Related topics

An item in a TreeView.

Syntax

[<oRef> =] new TreeItem(<parent> [,<name expC>])

<oRef>

A variable or property—typically of <parent>—in which to store a reference to the newly created TreeItem object.

<parent>

The parent object—a TreeView object for top-level items, or another TreeItem—to which you're binding the TreeItem object.

<name expC>

An optional name for the TreeItem object. If not specified, the TreeItem class will auto-generate a name for the object.

Properties

The following tables list the properties and methods of interest in the TreeItem class. (No particular events are associated with this class.)

Property	Default	Description
baseClassName	TREEITEM	Identifies the object as an instance of the TreeItem class
bold	false	Whether the text label is bold
checked	false	Whether the item is visually marked as checked
className	(TREEITEM)	Identifies the object as an instance of a custom class. When no

		custom class exists, defaults to baseClassName
expanded	false	Whether the item's children are shown or hidden
firstChild		The first child tree item
handle		The Windows tree item handle (similar to *hWnd*)
image		Image displayed between checkbox and text label when item does not have focus
level		The tree level of the item
nextSibling		The next tree item with the same parent
noOfChildren		The number of child tree items
prevSibling		The previous tree item with the same parent
selectedImage		Image displayed between checkbox and text label when item has focus
text		The text label of the tree item

Method	Parameters	Description
ensureVisible( )		Expands the tree and scrolls the tree view if necessary to make the tree item visible
select( )		Makes the tree item the selected item in the tree
setAsFirstVisible( )		Expands the tree and scrolls the tree view if necessary to try to make the tree item the first (topmost) visible tree item
sortChildren( )		Sorts the child tree items

The following table lists the common properties, events, and methods of the TreeItem class:

Property		Event		Method
name parent		beforeRelease		release( )

Description

Each item in a tree view can contain text, an icon image that can change when the item is selected, and a checkbox. You can replace the checkbox images with a different pair of images to represent any two-state condition.

A TreeItem object can contain other TreeItem objects as child objects in a subtree, which can be expanded or collapsed.

class TreeView

Topic group Related topics

An expandable tree.

Syntax

[<oRef> =] new TreeView(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created TreeView object.

<container>

The container—typically a Form object—to which you're binding the TreeView object.

<name expC>

An optional name for the TreeView object. If not specified, the TreeView class will auto-generate a name for the object.

Properties

The following tables list the properties, events, and methods of interest in the TreeView class.

Property	Default	Description
allowDrop	false	Whether dragged objects can be dropped on the TreeView
allowEditLabels	true	Whether the text labels of the tree items are editable
allowEditTree	true	Whether items can be added or deleted from the tree
anchor	0 – None	How the TreeView object is anchored in its container (0=None, 1=Bottom, 2=Top, 3=Left, 4=Right, 5=Center, 6=Container)
baseClassName	TREEVIEW	Identifies the object as an instance of the TreeView class
checkBoxes	true	Whether each tree item has a checkbox
checkedImage		The image to display when a tree item is checked instead of a checked check box
className	(TREEVIEW)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorNormal	WindowText /Window	The color of the text labels and background
disablePopup	false	Whether the tree view's popup menu is disabled
firstChild		The first child tree item
firstVisibleChild		The first TreeItem that is visible in the TreeView area
hasButtons	true	Whether + and - icons are displayed for TreeItems that have children
hasLines	true	Whether lines are drawn between TreeItems
image		Default image displayed between checkbox and text label when a TreeItem does not have focus
imageScaleToFont	true	Whether TreeItem images automatically scale to match the text label font height
imageSize		The height of TreeItem images in pixels
indent		The horizontal indent, in pixels, for each level of TreeItems
linesAtRoot	true	Whether a line connects the TreeItems at the first level
selected		The currently selected TreeItem
selectedImage		Default image displayed between CheckBox and TextLabel when a TreeItem has focus
showSelAlways	true	Whether to highlight the selected item in the tree even when the TreeView does not have focus
toolTips	true	Whether to display TextLabels as tooltips if they are too long to display fully in the TreeView area as the mouse passes over them
trackSelect	true	Whether to highlight and underline TreeItems as the mouse passes over them
uncheckedImage		The image to display when a TreeItem is not checked instead of an empty check box

Event	Parameters	Description
canChange		Before selection moves to another TreeItem; return value determines if selection can leave current TreeItem
canEditLabel		When attempting to edit text label; return value determines whether editing is allowed
canExpand	<oItem>	When attempting to expand or collapse a TreeItem; return value determines whether expand/collapse occurs
onChange		After the selection moves to another TreeItem
onCheckBoxClick		After a checkbox in a TreeItem is clicked
onDragEnter	<left expN> <top expN> <type expC> <name expC>	When the mouse enters the TreeView display area during a Drag&Drop operation
onDragLeave		When the mouse leaves the TreeView display area without having dropped an object
onDragOver	<left expN> <top expN> <type expC> <name expC>	While the mouse drags an object over the TreeView display area during a Drag&Drop operation
onDrop	<left expN> <top expN> <type expC> <name expC>	When the mouse button is released over the TreeView display area during a Drag&Drop operation
onEditLabel	<text expC>	After the TextLabel in a TreeItem is edited; may optionally return a different label value to save
onExpand	<oItem>	After a TreeItem is expanded or collapsed

Method	Parameters	Description
count( )		Returns the total number of TreeItems in the tree
getItemByPos( )	<col expN> <row expN>	Returns an object reference to a TreeItem object located at a specified position
loadChildren( )	<filename expC>	Loads and instantiates child TreeItems from a text file
releaseAllChildren( )		Deletes all TreeItems in the tree
sortChildren( )		Sorts child TreeItems
streamChildren( )	<filename expC>	Streams child TreeItems out to a text file
visibleCount( )		Returns the number of TreeItems visible in the tree view area

The following table lists the common properties, events, and methods of the TreeView class:

Property		Event		Method
before	hWnd	onClose	onMiddleDblClick	drag( )
borderStyle	id	onDesignOpen	onMiddleMouseDown	move( )
dragEffect	left	onDragBegin	onMiddleMouseUp	release( )
enabled	mousePointer	onGotFocus	onMouseMove	setFocus( )
fontBold	name	onHelp	onOpen	
fontItalic	pageno	onLeftDblClick	onRightDblClick	
fontName	parent	onLeftMouseDown	onRightMouseDown	
fontSize	printable	onLeftMouseUp	onRightMouseUp	
fontStrikeout	statusMessage	onLostFocus		
fontUnderline	systemTheme			
form	speedTip			

height
helpFile
helpId
tabStop
top
visible
width

Description

A TreeView displays a collapsible multi-level tree. There are four ways to create the tree:

- Explicitly add items with code, like the code generated by the Form Designer.
- Interactively through the TreeView's runtime user interface (right-clicking or pressing certain keys).
- Data-driven code that reads a table and dynamically creates the tree items.
- Use the TreeView's *loadChildren*( ) method to add items previously saved to a text file using *streamChildren*( ).

The TreeView object acts as the root of the tree. It contains the first level of TreeItem objects, which can contain their own TreeItem objects, thus forming a tree.

Unlike the deeper levels of the tree, you cannot collapse the first level of a tree. Therefore, you may want to use only one item at the first level of the tree to make the entire tree collapsible.

abandonRecord( )

Topic group Related topics Example

Abandons changes to the current record.

Syntax

<oRef>.*abandonRecord*( )

<oRef>

An object reference to the form.

Property of

Form, SubForm

Description

Use *abandonRecord*( ) for form-based data handling with tables in work areas. When using the data objects, *abandonRecord*( ) has no effect; use the rowset's *abandon*( ) method instead.

Form-based data buffering lets you manage the editing of existing records and the appending of new records. The temporary record created by *beginAppend*( ) and editing changes to the current record are not written to the table until there is navigation off the record, or until *saveRecord*( ) is called. Each work area has its own separate edit buffer. For example, if you *beginAppend*( ) in two separate work areas, you must call *abandonRecord*( ) while each work area is selected to abandon the changes.

Before calling *abandonRecord*( ), you can use *isRecordChanged*( ) to determine if changes have been made. If so, you may want to confirm the action before proceeding.

activeControl

Topic group Related topics

Contains a reference to the object that currently has focus.

Property of

Form, SubForm

Description

Use the *activeControl* property to reference the object that currently has focus.

An object gets focus in three ways:

The user tabs to the object.
The user clicks the object.
The *setFocus*( ) method of the object is executed.

Note

When a PushButton's *speedBar* property is set to true, and therefore a PushButton cannot get focus, the form's *activeControl* property does not show the PushButton as being the current activeControl.

alias

Topic group Related topics

Determines the table that is accessed by a Browse object.

Property of

Browse

Description

Use the *alias* property to identify a table to display in a browse object. For example, when the form is based on a .QBE query that opens two or more files in a parent-child relation, you use *alias* to determine which table appears in the Browse object.

Aliases are used for tables open in work areas, not data objects. When the form uses data objects, use a Grid control, which can *dataLink* directly to a Rowset object.

alignHorizontal

Topic group Related topics

Determines the horizontal alignment of text in a Text or TextLabel component.

Property of

Text, TextLabel

Description

alignHorizontal determines the way the text displays within the horizontal plane of its rectangular frame. Set it to one of the following:

Value	Alignment
0	Left
1	Center
2	Right
3	Justify (Text component only)

alignment [Image]

Topic group Related topics

Determines the size and position of the graphic inside an Image object.

Property of

Image

Description

If a graphic is smaller than the Image object that displays it, it can be stretched to fill the Image object or positioned inside the Image object with empty space around it. Assign one of the following settings to the *alignment* property of an Image object to determine how the graphic is aligned.

Setting	Description
0 (Stretch)	Enlarge graphic to fill the entire Image object
1 (Top Left)	In the top left corner of the Image object
2 (Center)	Centered in the Image object
3 (Keep Aspect Stretch)	Maintains the original height/width (aspect) ratio when stretching the graphic until it fills either dimension of the Image object
4 (True Size)	No changes to the graphic

If the graphic is larger than the Image object, both Stretch and Keep Aspect Stretch will reduce the graphic to fit the Image object so that the entire image is visible. Top Left and Center will both display whatever fits in the Image object.

True Size does not change the graphic at all. The Image object is dynamically resized to display the graphic. This is the fastest option, because *dBASE Plus* doesn't have to do any stretching or shrinking.

alignment [Text]

Topic group Related topics

Positions text in a Text object.

Property of

Text

Description

The Text object's *alignment* property is maintained primarily for compatibility. It is an enumerated property that can have the following values:

Setting	Description
0 (Top Left)	Adjacent to the top edge and the left edge
1 (Top Center)	Adjacent to the top edge and centered horizontally
2 (Top Right)	Adjacent to the top edge and the right edge
3 (Center Left)	Centered vertically and adjacent to left edge
4 (Center)	Centered horizontally and vertically
5 (Center Right)	Centered vertically and adjacent to right edge
6 (Bottom Left)	Adjacent to the bottom edge and the left edge
7 (Bottom Center)	Centered horizontally and adjacent to the bottom edge

8 (Bottom Right)	Adjacent to the bottom edge and the right edge
9 (Wrap Left)	Same as Top Left
10 (Wrap Center)	Same as Top Center
11 (Wrap Right)	Same as Top Right

The Text object's *alignHorizontal* and *alignVertical* properties control the alignment in the horizontal and vertical plane separately. They also include options to justify the text. When either property is set, the *alignment* property is also changed to match (justify becomes top or left). When the *alignment* property is set, the other two properties are changed to match.

Text wrapping is controlled by the *wrap* property.

alignVertical

Topic group Related topics

Determines the vertical alignment of text in a Text or TextLabel component.

Property of

Text, TextLabel

Description

alignVertical determines the way the text displays within the vertical plane of its rectangular frame. Set it to one of the following:

Value	Alignment
0	Top
1	Middle
2	Bottom
3	Justify (Text component only)

allowAddRows

Topic group Related topics

Whether rows can be added directly through a Grid object.

Property of

Grid

Description

allowAddRows determines whether moving down past the last row in a grid starts the append of a new row. It has no control over adding row in other ways, like by calling the rowset's *beginAppend*( ) method. If the rowset switches to Append mode, the grid will synchronize itself with the rowset and display the new row.

Set *allowAddRows* to *false* to prevent the accidental appending of new rows when navigating through a grid.

allowColumnMoving

Topic group Related topics

Whether the user may rearrange the columns in a grid with the mouse.

Property of

Grid

Description

By default, *allowColumnMoving* is *true*, which means that the user can rearrange the columns in a grid by clicking and dragging the column headings. Set *allowColumnMoving* to *false* to disable this ability. Since rearranging columns requires column headings, *hasColumnHeadings* must be set to *true* for *allowColumnMoving* to have any affect.

allowColumnSizing

Topic group Related topics

Whether the user may resize the columns in a grid with the mouse.

Property of

Grid

Description

By default, *allowColumnSizing* is *true*, which allows the user to resize columns in a grid by clicking and dragging between the column headings. Set *allowColumnSizing* to *false* to disable this feature. Since resizing columns requires column headings, *hasColumnHeadings* must be set to *true* for *allowColumnSizing* to have any affect.

allowDrop

Topic group Related topics

For Drag&Drop operations; determines if an object will allow dragged objects to be dropped on it.

Property of

Browse, Container, Form, Grid, Image, ListBox, NoteBook, PaintBox, ReportViewer, SubForm,TreeView

Description

Set the *allowDrop* property to *true* when you want to enable an object's ability to be a Drop Target. The default is *false*.

An object becomes an active Drop *Target* when its *allowDrop* property is set to true. Similarly, an object becomes an active Drop *Source* when the value of its *dragEffect* property is set greater than 0. Except for forms, all Drop Target objects may also be Drop Sources.

allowEditing

Topic group Related topics

Whether a grid is read-only.

Property of

Grid

Description

By default, *allowEditing* is *true*. Set *allowEditing* to *false* to make the grid read-only.

allowEditLabels

Topic group Related topics

Whether the text labels of the tree items are editable.

Property of

TreeView

Description

When *allowEditLabels* is *true*, the user can edit the *text* of the tree items in a tree by pressing F2 or clicking on a tree item a second time with a tree item selected.

Set *allowEditLabels* to *false* to prevent the user from editing all the items in a tree, or use the *canEditLabel* event to conditionally allow or prevent editing.

allowEditTree

Topic group Related topics

Whether items can be added or deleted from the tree.

Property of

TreeView

Description

When *allowEditTree* is *true*, the user can add another leaf node to a tree item by pressing Ins, or delete a tree item by pressing Del when the tree item is selected.

Set *allowEditTree* to *false* to prevent the user from inserting or deleting items in the tree.

allowFullScrollRange

Topic group Related topics Example

Can be used to prevent a grid's vertical scrollbar from being used to jump to the bottom of a rowset (via rowset.last()) due to dragging the scrollbar's thumb to the bottom of its range or when clicking the scrollbar near the bottom of its range. Instead, the grid will perform a Page Down and position the scrollbar thumb near the middle of its range.

Property of

Grid

Description

When set to True, the default, allowFullScrollRange allows the grid to scroll to the last row of a table when the scrollbar is dragged to the bottom of its range.

When set to False, allowFullScrollRange limits the grid to navigating forward one page full of rows when the scrollbar is dragged to the bottom of its range. When released the scrollbar thumb *bounces* back to the middle position of the scroll range.

Set allowFullScrollRange to False if you are datalinking a grid to a very large database table from a DBMS server containing millions or rows or more.

This will prevent inadvertently triggering a call to rowset.last() which can result in a long delay and or cause dBASE to become unresponsive while the server attempts to send millions of rows to the workstation.

allowRowSizing

Topic group Related topics

Whether the user may resize the rows in a grid with the mouse.

Property of

Grid

Description

By default, *allowRowSizing* is *true*, which means that the user can resize the rows in a grid by clicking and dragging between the row indicator in the left column. Set *allowRowSizing* to *false* to disable this ability.

alwaysDrawCheckBox

Topic group Related topics

Determines if a columnCheckBox control is painted with a checkbox for all checkBox cells in the Grid.

Property of

Grid

Description

By default, *alwaysDrawCheckBox* is set to *true*, the checkbox is drawn for all checkBox cells. When *alwaysDrawCheckBox* is set to *false,* the checkbox is only drawn in a cell if it has focus.

anchor

Topic group Related topics

Specifies whether an object stays in the same relative position when its container is resized.

Property of

ActiveX, Browse, Container, Editor, Image, Grid, NoteBook, OLE, ReportViewer, TabBox, Text, TreeView

Description

Use *anchor* to specify whether a control should maintain its location and resize itself to match its container, which is usually the form. Anchored controls claim space in the order in which they

are created (the z-order). Once an anchored control claims space in its container, that space cannot be used by another anchored control.

anchor is an enumerated property and, with the exception of TabBox controls, accepts the following values:

Value	Description
0	None, do not anchor
1	Bottom
2	Top
3	Left
4	Right
5	Center
6	Container

When anchored to the bottom or top, the *width* of the control resizes to match the width of its container. When anchored to the left or right, the *height* of the control resizes to match the height of its container. Center and container anchors behave identically: the control fills the center of the container, sizing itself to fill all the space not claimed by another anchored control; if there are no other anchored controls in the container, the control resizes to fill its container.

When used with TabBox controls, the *anchor* property accepts only these values,

0 - Do not anchor
1 - Bottom

append

Topic group Related topics

Whether records can be added directly through a Browse object.

Property of

Browse

Description

append determines whether moving down past the last record in a browse starts the append of a new record. It has no control over adding records in other ways, like by calling the form's *beginAppend*( ) method. If a new record is added, the browse will show it.

Set *append* to *false* to prevent the accidental appending of new records when navigating through a browse.

appSpeedBar

Topic group Related topics

Determines whether the built-in toolbar is displayed while the form has focus.

Property of

Form

Description

Use the Form's *appSpeedBar* property to hide or display the Standard Toolbar when a form receives focus.

Value	Mode
0	Hide
1	Display
2	Use the current _app object's speedBar setting.

The Form's *appSpeedBar* property does not change the _app object's current speedBar setting. Instead, when *appSpeedBar* is set to 0 or 1, it overrides _app.speedBar when the form receives focus. To change the _app object's speedBar setting *See* *speedBar [_app]*. The default setting for *appSpeedBar* is 2.

Setting form.appSpeedBar to 0 (zero), hides the built-in toolbars while a form has focus. Any application toolbars (toolbars built from class Toolbar) that are attached to the form will display if they are not hidden.

Setting form.appSpeedBar to 1 allows the built-in toolbars to be displayed while a form has focus. If there are any application toolbars attached to the form they will display as well unless they are hidden.

autoCenter

Topic group Related topics

Determines if a form is automatically centered when it is opened.

Property of

Form, SubForm

Description

Use *autoCenter* to automatically center a form when it is opened. If *autoCenter* is *true*, MDI forms are centered in the MDI frame window; SDI forms are centered on-screen. If *autoCenter* is *false*, the form is positioned according to its *top* and *left* properties.

autoDrop

Topic group Related topics

Determines if the drop-down list drops automatically when the combobox gets focus.

Property of

ComboBox

Description

Set *autoDrop* to *true* to make the drop-down list portion of a combobox drop automatically when the combobox gets focus. Whenever the combobox loses focus, its drop-down list is always closed, no matter how it was opened.

autoDrop has no effect when the *style* of the combobox is Simple (0).

autoSize

Topic group Related topics

Determines if a form is automatically sized to contain its objects when the form is opened.

Property of

Form, SubForm

Description

Use *autoSize* to determine how a form is sized and proportioned.

If you set the *autoSize* property of a form to *true*, the form is automatically adjusted to contain its objects when it is opened. If you set *autoSize* to *false*, the form assumes its assigned *height* and *width* when it is opened.

When you set the *autoSize* property of a form to *true*, the default dimensions are ignored. The user can still move or resize the form, but if the form is closed and reopened it is automatically resized again to contain its objects.

autoTrim

Topic group Related topics

Controls whether or not trailing spaces are trimmed from character strings loaded from the control's dataSource.

Property of

columnComboBox, ComboBox

Description

When True, enables automatic trimming of trailing spaces from option strings as they are loaded into a combobox's dropdown list in the following circumstances:

- the combobox is datalinked to a field object that has a lookupSQL and/or lookupRowset defined.

- the combobox's datasource specifies a FIELD in a table

The default for autoTrim is False.

background

Topic group Related topics

A form’s background image.

Property of

Form, SubForm

Description

Set the *background* property to the file name of a bitmap you want tiled in the background of your form. See class Image for the list of bitmap formats supported by *dBASE Plus*.

You may use any *dBASE Plus*-supported bitmap format.

Setting a background image supersedes the background color designated by the form's *colorNormal* property.

before

Topic group Related topics Example

The next object in the z-order of the form.

Property of

All form components and menus

Description

An object's *before* property contains a reference to the next object in the z-order, in other words, the object the current object comes before. The z-order is the order in which controls are created on the form. It is the same order in which they are created; the same order as they are listed in the .WFM file. If objects overlap, the one that is later in the z-order is drawn on top, with the exception of Line and Shape objects, which are always drawn on the form surface.

The form's *first* property contains a reference to the first control in the z-order. The z-order is a closed loop. The before property of the last control in the z-order points back to the first control.

The form's tab order is related to the z-order. The objects are in the same order, but only those objects that can receive focus are in the tab order. All visual components in the form are somewhere in the z-order. Non-visual components, such as Query objects, are not in the z-order.

You must reorder the objects in the form's class definition, by editing the code in the .WFM file or using the Form designer to visually reorder the objects.

beforeCellPaint

Topic group Related topics Example

An event fired just before a grid cell is painted.

Parameters

<bSelectedRow>

bSelectedRow is *true* if the grid cell being painted is part of a selected row. Otherwise bSelectedRow is *false*

Property of

ColumnCheckBox, ColumnComboBox, ColumnEditor, ColumnEntryField, ColumnHeadingControl, ColumnSpinBox

Description

Use the *beforeCellPaint* event to change the settings of a GridColumn's *editorControl* or *headingControl* just before the control is used to paint a grid cell.

After the grid cell has been painted, the *onCellPaint* event will fire. You must use the *onCellPaint* event to set the control back to it's prior, or it's default, state. Otherwise, the changes made in the *beforeCellPaint* event will affect the other cell's within the same grid column.

Using *beforeCellPaint*

In order to use *beforeCellPaint*, a grid must be created with explicitly defined GridColumn objects (accessible through the grid's columns property).

In a *beforeCellPaint* event handler, you can change an editorControl's or headingControl's properties based (optionally) on the current value of the cell. Within *beforeCellPaint*, the current cell value is contained in this.value.

Initializing a Grid that uses *beforeCellPaint*

When a form opens, a grid on the form is usually painted before the code setting up any *beforeCellPaint* event handlers is executed. Therefore, you should call the grid's *refresh*( ) method from the form's *onOpen* event to ensure the grid is painted correctly when the form opens.

Warning: The grid's painting logic is optimized to only load an editorControl's value when it needs to paint it, or give it focus. This means the value loaded into other column's editorControls may not be from the same row as the one used for the currently executing *beforeCellPaint* event. You should instead, therefore, use the values from the appropriate rowset field objects in order to ensure you are using values from the correct row.

beforeCloseUp

Topic group Related topics

Fires just before dropdown list is closed for a style 1 or 2 combobox.

Parameters

none

Property of

ComboBox

Description

beforeCloseUp fires just before the dropdown list is closed for a style 1 or 2 combobox.

It can be used, in combination with beforeDropDown, to track when a combobox's dropdown list is open or closed.

beforeRelease

Topic group

fires before the object has been released and is about to be destroyed.

Property of

Most dBASE form and data objects.

Description

Use beforeRelease to perform any extra manual cleanup, if necessary, before an object is released. beforeRelease fires when an object is about to be destroyed.

beforeRelease will fire under the following conditions:

- When calling the release() method of an object
- When issuing the RELEASE command
- When an object is run without being assigned to a memVar then closing the object. This will destroy the object from memory causing the the beforeRelease event to fire.
- when using a memVar that is assigned to one of these objects and subsequently releasing the memVar. Simply closing the object in this instance does not fire beforeRelease.
  The beforeRelease event will fire in this case only when the memVar itself is destroyed either by using the RELEASE command, when the application is closed, or any other circumstance that results in the memVar being released from memory.

The order in which beforeRelease() fires for a Form and its contained objects is not strictly in the order in which these objects are released. In some cases, beforeRelease() will fire sooner than an object's actual release when it is notified by its parent object or by its associated datamodule, database, query, or storedproc object that it will soon be released.

The order in which objects on a form are actually released has not been changed from earlier versions of dBASE Plus except for subforms. Subform release has been moved from near the end of the release process up to the beginning of the release process.

One other change made is that removal of a form's code from memory has been moved later in the release process so that it occurs after all objects on the form have been released.
This is to prevent CLASS NOT FOUND errors that would otherwise occur if an object's beforeRelease property is set to a method of its form.

Here is the firing order for beforeRelease() when a form (or subform) is being released:

1. Subforms (if any)
2. Form Components contained in the form's elements array property (this includes any Contaner and Notebook objects and their contained objects)
3. Menubar assigned to Form's menufile property
4. Popup assigned to Form's popupmenu property
5. Form
6. Database (parented by the form). Each Database object (in turn) notifies any associated Query and StoredProc objects which in turn, notify their Rowset objects.
7. Any other objects assigned to properties or custom properties of the form and that are not in the form's elements array property. This includes:
   - Query or StoredProc objects parented directly to a form and NOT assigned to a database object
   - Datamodule objects on the form
   - Session objects on the form
   - Popup and menu objects parented by the form
   - Toolbar objects parented by the form
   - Arrays and other non-visual objects parented by the form

beforeDropDown

Topic group Related topics

Fires just before dropdown list is opened for a style 1 or 2 combobox.

Parameters

none

Property of
ComboBox

Description
beforeDropDown fires just before the dropdown list is opened for a style 1 or 2 combobox.
It can be used, in combination with beforeCloseUp, to track when a combobox's dropdown list is open or closed.

beforeEditPaint

Topic group Related topics

For a style 0 or 1 combobox, fires for each keystroke that modifies the value of the combobox, just before the new value is displayed

Parameters
none

Property of
ComboBox

Description
For a style 0 or 1 combobox, fires for each keystroke that modifies the value of the combobox, just before the new value is displayed

beginAppend( )

Topic group Related topics

Creates a temporary buffer in memory for a record that is based on the structure of the current table, letting the user input data to the record without automatically adding the record to the table.

Syntax

<oRef>.*beginAppend*( )

<oRef>

An object reference to the form.

Property of

Form, SubForm

Description

Use *beginAppend*( ) for form-based data handling with tables in work areas. When using the data objects, *beginAppend*( ) has no effect; use the rowset's *beginAppend*( ) method instead.

*beginAppend( )*creates a single record buffer in the current table, without actually adding the record to the table until *saveRecord( )* is issued. While this buffer exists, the user can input data to the record with controls such as an entry field or a check box. Use *saveRecord( )* to append the record to the currently active table, and use *abandonRecord( )* to discard the record. Calling *beginAppend*( ) instead of *saveRecord*( ) will write the new record and empty the buffer again so

you can add another record. Use *isRecordChanged( )* to determine if the record has been changed since the *beginAppend( )*was issued.

When appending records with *beginAppend*( ) the new record will not be saved when you call *saveRecord*( ) unless there have been changes to the record; the blank new record is abandoned. This prevents the saving of blank records in the table. (If you want to create blank records, use APPEND BLANK). You can check there have been changes by calling *isRecordChanged*( ). If *isRecordChanged*( ) returns *true*, you should validate the record with form-level or row-level validation before writing it to the table.

Using *beginAppend( )* has different results than using either BEGINTRANS( ) and APPEND BLANK or APPEND AUTOMEM. With these commands, if you cancel the append operation, you have a record marked as deleted added to the table. If you use *abandonRecord( )* to cancel the *beginAppend( )* operation, a new record is never added to the table.

bgColor

Topic group Related topics

The background color of data displayed in grid cells, as well as the empty area to the right of the last column and below the last grid row.

Property of

Grid

Description

You may specify any single color for the background color. For a list of valid colors, see *colorNormal*.

The effect of the *bgColor* property on grid cells can be overridden by the background color specified in the Grid object's *colorNormal* property by setting it to a valid non-null string. In addition, the *bgColor* property can be overridden by setting the *bgColor* property of a GridColumn's *editorControl* to a valid non-null string.

The *bgColor* property defaults to "gray".

bitmapAlignment

Topic group Related topics

Specifies the arrangement of bitmap and text, on a pushButton, when both exists.

Property of

PushButton

Description

Supported options include:

0 Default.

1 Left

2 Top

3 Right

4 Bottom

When the *bitmapAlignment* property is set to 0, Default, the bitmap is positioned as follows:

If the pushButton does not contain text, the bitmap is centered.

If the pushButton contains both text and a bitmap, the text is positioned according to the setting of the *textLeft* property.

If the *textLeft* property is set to *false*, the text is positioned to the right, and the bitmap to the left.

If the *textLeft* property is set to *true*, the text is positioned to the left, and the bitmap to the right.

When the *bitmapAlignment* property is set to 1, 2, 3, or 4, it overrides the *textLeft* property's setting as follows:

1	Left	Positions the bitmap to the left, and any text to the right.
2	Top	Positions the bitmap at the top, and any text at the bottom.
3	Right	Positions the bitmap to the right, and any text to the left.
4	Bottom	Positions the bitmap at the bottom, and any text at the top.

Additional considerations:

When the *textLeft* property is overridden, by setting the *bitmapAlignment* property to 1, 2, 3, or 4, it will still affect the alignment when the text occupies more than one line, as follows:

When the *textLeft* property is set to *true*, the text lines are left aligned

When the *textLeft* property is set to *false*, the text lines are centered.

The text and bitmap may overlap when using a setting other than 0 – Default.

bold

Topic group Related topics

Whether the text of an object is bold.

Property of

TreeItem

Description

Set *bold* to *true* to make the text of a TreeItem object bold.

border

Topic group Related topics

Determines whether an object is surrounded with a border.

Property of

Editor, Entryfield, OLE, Rectangle, SpinBox, Text

Description

The *border* property is maintained primary for compatibility. Every object that has a *border* property also has a *borderStyle* property. One of the choices for *borderStyle* is None, while *border* can be either *true* or *false*. Both these properties apply.

If you pick an actual border with *borderStyle*, *border* determines whether that border is displayed. If you choose the None *borderStyle*, no border will appear, even if *border* is *true*.

borderStyle

Topic group Related topics

Determines the border around the object.

Property of

Most form components

Description

borderStyle determines the display style of an object's rectangular frame. Set it to one of the following:

Value	Style
0	Default
1	Raised
2	Lowered
3	None
4	Single
5	Double
6	Drop shadow
7	Client
8	Modal
9	Etched in
10	Etched out

The border is drawn inside the bounds of the object; therefore for thick borders like Drop shadow, there is noticeably less space in the object for the actual contents.

The border is not drawn if *border* is *false*. If *borderStyle* is None, no border appears even if *border* is *true*.

bottom

Topic group Related topics

The vertical position of the lower end of a Line object.

Property of

Line

Description

Use the *bottom* property in combination with the *right*, *left*, and *top* properties to determine the position and length of a line object.

The unit of measurement in a form or report is controlled by its *metric* property. The default metric for forms is characters, and for reports it's twips.

buttons

Topic group Related topics

Whether a notebook's tabs appear as buttons

Property of

NoteBook

Description

Set *buttons* to *true* if you want the notebook tabs to appear as separate buttons instead of tabs attached to the notebook page.

canChange

Topic group Related topics Example

Event fired before selection moves to another tree item; return value determines if selection can leave current tree item.

Parameters

none

Property of

TreeView

Description

Use *canChange* to prevent focus from moving to another item in the tree unless certain conditions are met. The *canChange* event handler can either return a value of *true*, which allows the focus to move, or *false*, which prevents the focus change.

The event handler usually uses the tree view's *selected* property to get the currently selected tree item. Note that if no tree item is selected, the property contains *null*, so your event handler must check for that. In particular, when deleting a tree item, the focus must move to another tree item, and the currently selected item has just been deleted, and therefore *selected* will be *null*.

canClose

Topic group Related topics Example

Event fired when an attempt is made to close the form; return value determines if the form can be closed.

Parameters

none

Property of

Form, SubForm

Description

Use *canClose* to prevent a form from closing until certain conditions are met.

The *canClose* event handler can either return a value of *true*, which allows the form to close, or *false*, which prevents the form from closing.

When a form is closed by calling *close*( ) or clicking the Close icon, pending changes in the data buffer are saved. When attempting to close, the form fires the current control's *valid* event, if

any, so there's no need to verify individual controls if they have *valid* event handlers. However, you should always perform form- or row-level validation to check controls that you have not visited.

canEditLabel

Topic group Related topics Example

Event fired when attempting to edit text label; return value determines if editing is allowed.

Parameters

none

Property of

TreeView

Description

Use *canEditLabel* to conditionally allow editing of a tree item's text label. The *canEditLabel* event handler can either return *true* to allow editing, or *false* to prevent it.

Set *allowEditLabels* to *false* to prevent all label editing. In that case, *canEditLabel* will never fire.

canExpand

Topic group Related topics Example

Event fired when attempting to expand or collapse a tree item; return value determines whether expand/collapse occurs.

Parameters

<oItem>

The TreeItem whose + or - has been clicked.

Property of

TreeView

Description

Use *canExpand* to conditionally allow the expansion or collapsing of a tree item's subtree. The *canExpand* event handler can either return *true* to allow the action, or *false* to prevent it.

The *canExpand* event only affects the user interface. You can still expand or collapse a tree item programmatically by setting the item's *expanded* property, in which case *canExpand* does not fire.

canNavigate

Topic group Related topics

Event fired when an attempt is made to navigate in a table; return value determines if the record pointer moves.

Parameters

<workarea expN>

The work area number where the navigation is attempted.

Property of

Form, SubForm

Description

The form's *canNavigate* event is used mainly for form-based data handling with tables in work areas. It also fires when attempting navigation in the form's primary rowset.

Use *canNavigate* to prevent navigation until certain conditions are met. Navigation saves pending changes in the data buffer, so you should call row- or form-level validation in the *canNavigate* to make sure data should be saved.

Because *canNavigate* fires while still on the current record, you may also use it to perform some action just before you leave. In this case, the *canNavigate* would always return *true* to allow the navigation.

When using tables in work areas, *canNavigate* will not fire unless the form is open and has controls *dataLink*ed to fields. For example, if you USE a table, create and open an empty Form, assign an *canNavigate* event handler, and SKIP in the table, the *canNavigate* will not fire simply because the form is open.

When attempting navigation in the form's primary rowset, the form's *canNavigate* fires before the rowset's *canNavigate*, and the <workarea expN> parameter is zero. If the form's *canNavigate* returns *false*, nothing further happens; the rowset's *canNavigate* does not fire, and no navigation occurs.

canSelChange

Topic group Related topics Example

Event fired before another NoteBook tab is selected; return value determines if the new tab is selected.

Parameters

<nNewSel expN>

The number of the tab about to be selected.

Property of

NoteBook

Description

Use *canSelChange* to prevent the user from selecting another NoteBook tab until certain conditions are met. The parameter passed by the event, nNewSel, is an integer value representing the number of the NoteBook tab to be selected. The tabs are numbered (beginning with 1) according to the order of the array elements in the NoteBook's *dataSource* property.

Because *canSelChange* fires while still on the current NoteBook tab, you may also use it to perform some action just before you allow the new tab to be selected. In this case, the *canSelChange* event handler would always return *true* to allow selection of the new tab.

cellHeight

Topic group Related topics

The height of each cell in the grid.

Property of

Grid

Description

The *cellHeight* property reflects the height of the cells in the body of the grid.

checkBoxes

Topic group Related topics

Whether each tree item has a checkbox.

Property of

TreeView

Description

When *checkBoxes* is *true*, each tree item has a checkbox to the left of the text label and optional icon image. This checkbox is linked to the tree item's *checked* property. Whenever a checkbox is checked or unchecked, the tree's *onCheckBoxClick* event fires.

Use the *checkedImage* and *uncheckedImage* image properties to specify alternate images instead of the standard checkbox. Set *checkBoxes* to *false* to hide and disable the checkboxes in the tree.

checked

Topic group Related topics

Whether the item is visually marked as checked.

Property of

TreeItem

Description

TreeItem objects may be visually checked and unchecked by the user, or by assigning a value to the *checked* property. If *checked* is *true*, the tree item's checkbox is checked, or its *checkedImage* is displayed. If *checked* is *false*, the tree item's checkbox is unchecked, or its *uncheckedImage* is displayed.

checkedImage

Topic group Related topics

The image to display when a tree item is checked instead of a checked check box.

Property of

TreeView

Description

Use *checkedImage* to display a specific icon instead of the standard checked checkbox for the tree items in the tree that are checked. *uncheckedImage* optionally specifies the icon to display

for tree items that are not checked. The tree must have its *checked* property set to *true* to enable checking; each tree item has a *checked* property that reflects whether the item is checked.

The *checkedImage* property is a string that can take one of two forms:

RESOURCE <resource id> <dll name>
specifies an icon resource and the DLL file that holds it.
FILENAME <filename>
specifies an ICO icon file.

classId

Topic group Related topics

The ID string of an ActiveX control.

Property of

ActiveX

Description

To use an ActiveX control in a form, set the *classId* property to the control's ID string.

clearTics( )

Topic group Related topics

Clears manually-set tic marks in a Slider object.

Syntax

<oRef>.*clearTics*(<expN>)

<oRef>

The Slider object whose tics to clear.

<expN>

A numeric value, or an expression which evaluates to a numeric value.

Property of

Slider

Description

Call *clearTics*( ) to clear all tic marks set by *setTic*( ). <expN> can be any expression which evaluates to a positive, negative or fractional numeric value (fractional values are truncated). If <expN> is zero, no tic marks are cleared. If it is non-zero, all manually set tic marks are cleared.

clientEdge

Topic group Related topics

Whether an object appears to have a beveled inside edge.

Property of

Form, SubForm

Description

Set *clientEdge* to *true* to bevel a form's inside edge.

close( )

Topic group Related topics

Closes a form.

Syntax

<oRef>.*close*([expX])

<oRef>

An object reference to the form to close.

<expX>

An optional value to be returned by a form opened with readModal( )

Property of

Form, SubForm

Description

Use *close*( ) to close an open form or report. Modal forms (forms opened using the *readModal( )* method) may optionally return a value to the calling form or program. The returned value may be of any data type.

When you try to close a form, *dBASE Plus* does the following:

1. Fires the *valid* event of the current object. If it returns a value of *false*, the form doesn't close.
    - Fires the *onLostFocus* event of the current object.
    - Fires the *canClose* event of the form. If it returns *false*, the form doesn't close.
    - Fires the *onLostFocus* event of the form.
    - Removes the form and the objects it contains from the screen.
    - Fires the *onClose* event of the form.
    - Removes the form from memory if there are no other object references pointing to the form.

When a form is closed with *close*( ) or by clicking the Close icon, any pending changes in the record buffer are saved, as if *saveRecord*( ) was called. Closing a form by pressing Esc abandons changes before closing (if *escExit* is *true*).

close( ) returns *false* if the form was not closed successfully. If the form definition is not removed from memory, you can open the form again with *open*( ).

colorColumnLines

Topic group Related topics

The color of the lines that separate the columns in a Grid object.

Property of

Grid

Default

silver

Description

The *colorColumnLines* property controls the color of the lines that separate columns in a Grid object. When the *colorColumnLines* property is set to null, the value of the *colorColumnLines* property will be restored to its default value. The color of the lines in a Grid Object's top and left headers is not affected by this property.

colorHighlight

Topic group Related topics

Sets the color of the object that has focus.

Property of

Browse, ColumnEditor, ComboBox, Editor, Entryfield, Grid, ListBox, SpinBox, TabBox

Description

Use the *colorHighlight* and *colorNormal* properties of an object so users can visually differentiate between an object that has focus, and one that doesn't. You may choose from the same color settings as the *colorNormal* property.

The *colorHighlight* of most controls defaults to an empty string, meaning it is colored no differently when it has focus. For this reason, you may set a particular color in the control's *colorNormal* property without having to override a default *colorHighlight* color as well.

For Grid objects, the *colorHighlight* property sets the text, and background, color for data displayed in a grid cell that has focus. It can be overridden by setting the *colorHighlight* property of a GridColumn's *editorControl* to a non-null value. The default setting for Grid objects is "WindowText/Window".

The color scheme in a TabBox is different. There, the *colorNormal* designates the color of the background behind the tabs, and the *colorHighlight* is the color of the selected tab. The unselected tabs are a fixed color, WindowText/Window.

colorNormal

Topic group Related topics Example

The color of an object.

Property of

Most form objects

Description

Use the *colorNormal* and *colorHighlight* properties of an object so users can differentiate visually when an object has focus and when it doesn't.

For some controls, in particular background colors, you specify a single color. For other controls, you specify two color settings with *colorNormal*: a foreground color (for text), and a background color. Color settings must be separated with a forward slash (/). Each color may be one of the following five color types, in any combination:

- Windows-named color
- Basic 16-color color code
- Hexadecimal RGB color triplet
- User-defined color name
- JavaScript color name

Color settings are not case-sensitive.

For Grid objects, the *colorNormal* property sets the text, and background, color for data displayed in grid cells that do not have focus. It can be overridden by setting the *colorNormal* property of a GridColumn's *editorControl* to a non-null value. The default setting for Grid objects is "WindowText/Window".

Windows-named colors are taken from the settings in the Display Properties. If the colors are changed in the Display Properties while a form is open, the form and any controls that use these values will change automatically. You can use any of the following Windows-named color settings for either the foreground or background color:

Color	Corresponding Display Properties Appearance Color
ActiveBorder	Active window border
ActiveCaption	Active title bar
AppWorkspace	Application background
Background	Desktop
BtnFace	3D objects
BtnHighlight	A shade lighter than 3D objects
BtnShadow	A shade darker than 3D objects
BtnText	3D objects font
CaptionText	Active title bar font
GrayText	preset gray—not available
Highlight	Selected items
HighlightText	Selected items font
InactiveBorder	Inactive window border
InactiveCaption	Inactive title bar
InactiveCaptionText	Inactive title bar font
InfoText	ToolTip
InfoBk	ToolTip font
Menu	Menu
MenuText	Menu font
Scrollbar	A shade lighter than 3D objects
Window	Window
WindowFrame	preset drop shadow—not available
WindowText	Window font

The following one- and two-letter basic color codes (with an optional + or * for brightness) are provided primary for compatibility with earlier versions of dBASE.

Color name	Foreground code	Background code
Black	N	N
Dark Blue	B	B
Green	G	G
Cyan	GB or BG	GB or BG

Dark Red	R	R
Purple	RB or BR	RB or BR
Brown	RG or GR	RG or GR
Light Gray	W	W
Dark Gray	N+	N*
Blue	B+	B*
Bright Green	G+	G+
Bright Cyan	GB+ or BG+	GB* or BG*
Red	R+	R*
Magenta	RB+ or BR+	RB* or BR*
Yellow	RG+ or GR+	RG* or GR*
White	W+	W*

Hexadecimal RGB (Red Green Blue) color triplets are expressed backwards in *dBASE Plus*; that is, Blue, Green, Red. You may specify one of approximately 16 million colors using a triplet. The color will be displayed as well as the settings of the display allow.

You may create your own RGB combinations and give them a name with the DEFINE COLOR command.

Finally, *dBASE Plus* supports JavaScript-standard color names. The following table lists those colors and their corresponding RGB values.

Color	Red	Green	Blue
aliceblue	F0	F8	FF
antiquewhite	FA	EB	D7
aqua	00	FF	FF
aquamarine	7F	FF	D4
azure	F0	FF	FF
beige	F5	F5	DC
bisque	FF	E4	C4
black	00	00	00
blanchedalmond	FF	EB	CD
blue	00	00	FF
blueviolet	8A	2B	E2
brown	A5	2A	2A
burlywood	DE	B8	87
cadetblue	5F	9E	A0
chartreuse	7F	FF	00
chocolate	D2	69	1E
coral	FF	7F	50
cornflowerblue	64	95	ED
cornsilk	FF	F8	DC
crimson	DC	14	3C

cyan	00	FF	FF
darkblue	00	00	8B
darkcyan	00	8B	8B
darkgoldenrod	B8	86	0B
darkgray	A9	A9	A9
darkgreen	00	64	00
darkkhaki	BD	B7	6B
darkmagenta	8B	00	8B
darkolivegreen	55	6B	2F
darkorange	FF	8C	00
darkorchid	99	32	CC
darkred	8B	00	00
darksalmon	E9	96	7A
darkseagreen	8F	BC	8F
darkslateblue	48	3D	8B
darkslategray	2F	4F	4F
darkturquoise	00	CE	D1
darkviolet	94	00	D3
deeppink	FF	14	93
deepskyblue	00	BF	FF
dimgray	69	69	69
dodgerblue	1E	90	FF
firebrick	B2	22	22
floralwhite	FF	FA	F0
forestgreen	22	8B	22
fuchsia	FF	00	FF
gainsboro	DC	DC	DC
ghostwhite	F8	F8	FF
gold	FF	D7	00
goldenrod	DA	A5	20
gray	80	80	80
green	00	80	00
greenyellow	AD	FF	2F
honeydew	F0	FF	F0
hotpink	FF	69	B4
indianred	CD	5C	5C
indigo	4B	00	82
ivory	FF	FF	F0
khaki	F0	E6	8C
lavender	E6	E6	FA
lavenderblush	FF	F0	F5

lawngreen	7C	FC	00
lemonchiffon	FF	FA	CD
lightblue	AD	D8	E6
lightcoral	F0	80	80
lightcyan	E0	FF	FF
lightgoldenrodyellow	FA	FA	D2
lightgreen	90	EE	90
lightgrey	D3	D3	D3
lightpink	FF	B6	C1
lightsalmon	FF	A0	7A
lightseagreen	20	B2	AA
lightskyblue	87	CE	FA
lightslategray	77	88	99
lightsteelblue	B0	C4	DE
lightyellow	FF	FF	E0
lime	00	FF	00
limegreen	32	CD	32
linen	FA	F0	E6
magenta	FF	00	FF
maroon	80	00	00
mediumaquamarine	66	CD	AA
mediumblue	00	00	CD
mediumorchid	BA	55	D3
mediumpurple	93	70	DB
mediumseagreen	3C	B3	71
mediumslateblue	7B	68	EE
mediumspringgreen	00	FA	9A
mediumturquoise	48	D1	CC
mediumvioletred	C7	15	85
midnightblue	19	19	70
mintcream	F5	FF	FA
mistyrose	FF	E4	E1
moccasin	FF	E4	B5
navajowhite	FF	DE	AD
navy	00	00	80
oldlace	FD	F5	E6
olive	80	80	00
olivedrab	6B	8E	23
orange	FF	A5	00
orangered	FF	45	00
orchid	DA	70	D6

palegoldenrod	EE	E8	AA
palegreen	98	FB	98
paleturquoise	AF	EE	EE
palevioletred	DB	70	93
papayawhip	FF	EF	D5
peachpuff	FF	DA	B9
peru	CD	85	3F
pink	FF	C0	CB
plum	DD	A0	DD
powderblue	B0	E0	E6
purple	80	00	80
red	FF	00	00
rosybrown	BC	8F	8F
royalblue	41	69	E1
saddlebrown	8B	45	13
salmon	FA	80	72
sandybrown	F4	A4	60
seagreen	2E	8B	57
seashell	FF	F5	EE
sienna	A0	52	2D
silver	C0	C0	C0
skyblue	87	CE	EB
slateblue	6A	5A	CD
slategray	70	80	90
snow	FF	FA	FA
springgreen	00	FF	7F
steelblue	46	82	B4
tan	D2	B4	8C
teal	00	80	80
thistle	D8	BF	D8
tomato	FF	63	47
turquoise	40	E0	D0
violet	EE	82	EE
wheat	F5	DE	B3
white	FF	FF	FF
whitesmoke	F5	F5	F5
yellow	FF	FF	00
yellowgreen	9A	CD	32

colorRowHeader

Topic group

Specifies the color of the indicator arrow or plus sign, and of the row header background.

Property of

Grid

Default

WindowText/BtnFace

Description

Use the *colorRowHeade*r property to set the color of the row header. The foreground color specifies the color of the indicator arrow or plus sign. The background color specifies the row header background color.

Values for this property are can be set using the Color Property Builder in the Form Designer. You can access this dialog box by clicking the wrench tool next to the *colorRowHeader* property in the Inspector.

For more information on the Color Property Builder, see the topic, Color Property Builder dialog box.

colorRowLines

Topic group Related topics

The color of the lines that separate the rows in a Grid object.

Property of

Grid

Default

silver

Description

The *colorRowLines* property controls the color of the lines that separate rows in a Grid object. When the *colorRowLines* property is set to null, the value of the *colorRowLines* property will be restored to its default value. The color of the lines in a Grid object's top and left headers is not affected by this property.

colorRowSelect

Topic group

Text and background color for visually selected rows of data.

Property of

Grid

Description

When the *rowSelect* property and/or the *multiSelect* property is true, the *colorRowSelect* property sets the text color and background color for a row of data that has been selected. The default for *colorRowSelect* is HighlightText/HighLight.

Values for this property are set in the Color Property Builder. You can access the dialog box by clicking the wrench tool next to the *colorRowSelect* property in the Inspector.

For more information on the Color Property Builder, see the topic, Color Property Builder dialog box.

columnCount

Topic group Related topics

The number of columns in the grid.

Property of

Grid

Description

columnCount is a read-only property that contains the number of columns in the grid; either the number of columns that are automatically created when no GridColumn objects are specified, or the number of GridColumn objects explicitly added.

columnNo

Topic group Related topics

The current position of the cursor in a line of text.

Property of
Editor

Description

Use the *columnNo* property to find the position of the cursor in the current line of text in an Editor window. When the Editor window is empty, *columnNo* will be 1.

The *columnNo* property can be used with the *lineNo* property to identify the character at the cursor by indexing into the contents of the Editor's *value* property.

The *columnNo* property is read-only.

columns

Topic group Related topics

An array of objects for each column in the grid.

Property of

Grid

Description

A grid's columns array contains explicitly created GridColumn objects, one for each column. If no GridColumn objects are created, the grid automatically creates columns, but the columns array is empty.

contextHelp

Topic group Related topics Example

Displays a context help question mark (?) next to the form or subform's close button.

Property of

Form, SubForm

Description

When *contextHelp* is set to true and mdi, maximize, and minimize are set to False, a button displaying a question mark (?) will display to the left of the form or subform's Close button in the top right portion of the form's title bar.

Clicking the mouse on the *contextHelp* button starts *contextHelp* mode which changes the mouse pointer and allows the user to click on a form component to trigger the component's onHelp() event. Note: clicking on the form itself will do nothing at this point.

onHelp() can be used to perform a context sensitive help lookup for the component.

The default for *contextHelp* is false.

copy( )

Topic group Related topics

Copies selected text to the Windows clipboard.

Syntax

<oRef>.*copy*( )

<oRef>

An object reference to the control from which to copy the text.

Property of

Browse, ComboBox, Editor, Entryfield, SpinBox

Description

Use *copy*( ) when the user has selected text and wants to copy it to the Windows clipboard.

When calling this method from a pushbutton's *onClick* event, the pushbutton should have its *speedBar* property set to *true*, so that it doesn't get focus. Otherwise, the edit control loses focus when the button is clicked, and there's nothing to copy.

If you have assigned a menubar to the form, you can use a menu item assigned to the menubar's *editCopyMenu* property instead of using the *copy*( ) method of individual objects on the form.

count( )

Topic group Related topics

Returns the number of prompts in a listbox, or the number of items in a tree.

Syntax

<oRef>.*count*( )

<oRef>

An object reference to the listbox or tree whose items to count.

Property of

ListBox, TreeView

Description

Use a listbox's *count*( ) method when you can't anticipate the number of prompts a listbox might have at runtime. For example, when you specify "FILE *.*" for the *dataSource* property, the number of prompts depends on the number of files in the current directory.

When using an array as the *dataSource* for a listbox, you can check the array's *size* property to get the number of items.

The tree view's *count*( ) method returns the total number of items in the entire tree, even if they are not displayed or hidden in a collapsed subtree. Use the *visibleCount*( ) method to count the items that are visible in the tree view.

cuaTab

Topic group Related topics

Determines cursor behavior when you press Tab while a control has focus.

Property of

Browse, Editor, Grid

Description

When *cuaTab* is *true*, pressing Tab moves to the next control in the form's tab order. When *cuaTab* is *false*, pressing Tab moves to the next field in a Grid or Browse object or moves the cursor to the next tab stop in an Editor object.

The same applies to pressing Shift+Tab, except that the movement is in reverse.

currentColumn

Topic group Related topics

The number of the column that has focus in the grid.

Property of

Grid

Description

Use the *currentColumn* property as an index into the *columns* array to refer to the GridColumn object that represents the column that currently has focus.

curSel

Topic group Related topics Example

Determines which prompt is selected in a component.

Property of

ListBox, NoteBook, TabBox

Description

Use *curSel* to get or set which prompt in a ListBox, NoteBook, or TabBox is highlighted. The prompts are determined by the component's *dataSource* property. The first prompt is prompt number 1.

Assigning a value to *curSel* fires the control's *onSelChange* event, as if the change was made manually.

cut( )

Topic group Related topics

Cuts selected text and places it on the Windows Clipboard.

Syntax

<oRef>.*cut*( )

<oRef>

An object reference to the control from which to cut the text.

Property of

Browse, ComboBox, Editor, Entryfield, SpinBox

Description

Use *cut*( ) when the user has selected text and wants to remove it from the edit control and place it on the Windows clipboard.

When calling this method from a pushbutton's *onClick* event, the pushbutton should have its *speedBar* property set to *true*, so that it doesn't get focus. Otherwise, the edit control loses focus when the button is clicked, and there's nothing to cut.

If you have assigned a menubar to the form, you can use a menu item assigned to the menubar's *editCutMenu* property instead of using the *cut*( ) method of individual objects on the form.

dataLink

Topic group Related topics Example

The Field object that is linked to the component.

Property of

Many form components

Description

You link a form component to a table's field by assigning a reference to the *dataLink* property of the component. The reference you assign is to the Field object that represents the field in an open query. This assignment is called *dataLink*ing. When a form component and Field object are linked in this way, they are said to be *dataLink*ed.

Exception: a Grid object is *dataLink*ed to a rowset, not a field.

For compatibility with earlier versions of dBASE, you may also assign the field name in a string. This technique is used for form-based data handling with tables open in work areas only.

Both field and component objects have a *value* property. (Fields in a table open in a work area do not have any properties, but they have a value; the concept is the same.) When they are *dataLink*ed, changes in one object's *value* property are echoed in the other. The form component's *value* property reflects the value displayed in the component at any given moment. If the component's value is changed, it is copied into the field after the component loses focus.

The *value* property for all fields in a rowset are set when you first open a query and updated as you navigate from row to row. The *value* properties for components *dataLink*ed to those fields are also updated at the same time, unless the rowset's *notifyControls* property is set to *false*. You can also force the components to be updated by calling the rowset's *refreshControls*( ) method, which is useful if you have set a field's *value* property through code.

Form-based data events such as *onNavigate* will not work unless the form has controls *dataLink*ed to fields. For example, if you USE a table, create and open an empty Form, assign an *onNavigate* event handler, and SKIP in the table, the *onNavigate* will not fire simply because the form is open.

The *dataLink* property is similar to the *dataSource* property used for Image objects, except that data displayed through the *dataLink* property can be changed, while data displayed through the *dataSource* property is always read-only.

A component's *dataLink* is automatically set when you use the Form wizard or use a field in the Field palette.

dataSource [options]

Topic group Related topics Example

The options that are displayed in a ComboBox, ListBox, NoteBook, or TabBox object.

Property of

ComboBox, ListBox, NoteBook, TabBox

Description

Use the *dataSource* property to set the options that are displayed in a ComboBox, ListBox, NoteBook, or TabBox object. The *dataSource* property for a ComboBox or ListBox is a string in one of the following five forms:

ARRAY <array>
: creates prompts from elements in an array object.

FIELD <field name>
: creates prompts from all the values in a field in a table file.

FILENAME [<filename skeleton>]
: creates prompts from file names in the current default directory that match the optional filename skeleton.

STRUCTURE
: creates prompts from all the field names in the currently selected table.

TABLES
: creates prompts from the names of all tables in the currently selected database. For the default database, this is all the .DBF and .DB files in the current directory.

For a NoteBook or TabBox, only the ARRAY *dataSource* is allowed. The *dataSource* string in general is not case-sensitive, except that if you specify a literal array, the array contents will appear as specified.

Adding elements to an array after it has been assigned as a component's *dataSource* may not automatically update the component's options. Files added to the directory after the *dataSource* property has been set to a file mask will not automatically appear either.

To update the *dataSource*, you need to reassign the *dataSource* property. In most cases, you can simply reassert the property by assigning its current value to itself. For example, if you had originally specified all the GIF files in the current directory, the *dataSource* property assignment would look like this:

```
with (this.fileCombobox)
dataSource = "FILENAME *.GIF"
endwith
```

To update the file list when you press an Update button on your form, the button's *onClick* would look like this:

```
function updateButton_onClick( )
form.fileCombobox.dataSource += ""
```

You don't have to specify what the *dataSource* string is again, since it's already contained in the *dataSource* property. The += operator adds an empty string to reassign the value, which reasserts the *dataSource*. This makes your code easier to maintain, since the *dataSource* string is specified in only one place.

When using a FIELD as the datasource string, you can use the fields

value:

```
form.rowset.fields["myfield"].value
```

Or a reference to a field object:

```
form.rowset.fields["myfield"]
```

When using an array in the *dataSource* string, you can use a literal array, for example,

```
array {"Chocolate", "Strawberry", "Vanilla"}
```

Or you can use a reference to an array object, for example,

```
array aFlavors
```

If you use a reference, that array must exist at the time the *dataSource* property is assigned. Since the *dataSource* property contains that string (in this example, array aFlavors), if you reassert the *dataSource* property as shown above, an updated version of the named array must exist. In this example, the array aFlavors must be accessible in the method updateButton_onClick( ).

For this reason, when using an updatable array as the *dataSource* property, the array is usually created as a property of the form. This makes the array equally accessible from the component that uses the array and from any other component that tries to reassert the *dataSource* property. In this example, the array aFlavors would be created as a property of the form, and the *dataSource* string would contain:

```
array form.aFlavors
```

The reference *form*.aFlavors is valid from the event handler of any component on the form.

dataSource [Image]

Topic group Related topics Example

The bitmap that is displayed in an Image object.

Property of

Image

Description

An Image object can display either a static file from disk, a resource image, or a bitmap stored in a table. Set the *dataSource* property to either one of the following:

- A string containing the word FILENAME, a space, and the name of a file. The string is not case-sensitive.
- A string of the form "RESOURCE <resource id> <dll name>", which specifies a bitmap resource and the DLL file that holds it.
- A string containing the form BINARY, a space, and the name of a binary field in a table open in a work area that contains bitmapped images.
- A reference to a field object in an open query that contains bitmapped images.

If you assign a field object (or a field in a work area) as the *dataSource*, the Image object will automatically update as you navigate from row to row, unless the rowset's *notifyControls* property is set to *false*.

The *dataSource* property is similar to the *dataLink* property used for Field objects, except that data displayed through the *dataLink* property can be changed, while data displayed through the *dataSource* property is always read-only.

An Image object's *dataSource* is automatically set when you use the Form Wizard or use a bitmap image field in the Field Palette.

default

Topic group Related topics

Determines if a pushbutton is the form's default pushbutton.

Property of

PushButton

Description

Use the *default* property to make a pushbutton the default pushbutton when the user submits a form by pressing Enter. This behavior is used primarily for dialog boxes, when either the OK or Cancel button being the default, whichever is more appropriate. Setting the *default* property of a pushbutton to *true* gives the pushbutton a visual highlight that identifies it as the default.

Setting the *default* property to *true* causes two things to happen when the user presses Enter when the focus is not on a pushbutton:

- The *onClick* subroutine of the default pushbutton executes.
- The *id* property of the default pushbutton is passed to the form's *onSelection* event handler.

However, if the user clicks on any pushbutton, the *onClick* event handler of that pushbutton executes. The *id* value of that pushbutton is passed to the *onSelection* event, even if the *default* property of another pushbutton is *true*.

If you give more than one pushbutton a *default* value of *true*, the last pushbutton to get the value is the default.

The *default* property will only work when SET CUAENTER is set to ON. When CUAENTER is OFF, the Enter key emulates the Tab key and merely shifts focus to the next control.

description

Topic group

A short description for an ActiveX control.

Property of

ActiveX

Description

An ActiveX object's *description* property contains a short description of the ActiveX control it represents. The description is provided by the control and is read-only.

designView

Topic group Related topics

Designates a .QBE query or table that is used when designing a form.

Property of

Form, SubForm

Description

Use *designView* to facilitate creating and *dataLink*ing a form that uses form-based data handling with tables in work areas. The value in *designView* is ignored at runtime. When using the data objects, do not use *designView* or *view*.

There are two main instances in which you may want to use *designView* instead of *view*.

If you know which tables will be open when the form is opened at runtime, use *designView* to avoid opening the tables again with *view* when the form is opened.

If you don't know which tables will be open when the form is opened at runtime, but need certain tables open to design the form, use *designView* to automatically open the tables at design time, regardless of the tables needed at runtime.

If you specify a *view* property for a form, you should not also specify a *designView* property.

disabledBitmap

Topic group Related topics

Specifies the graphic image to display in a pushbutton when the pushbutton is disabled.

Property of

PushButton

Description

Use *disabledBitmap* to indicate visually when a pushbutton is not available for use. A pushbutton is disabled when its *enabled* property is set to *false*.

The *disabledBitmap* setting can take one of two forms:

RESOURCE <resource id> <dll name>
specifies a bitmap resource and the DLL file that holds it.

FILENAME <filename>
specifies a bitmap file. See class Image for a list of bitmap formats supported by *dBASE Plus*.

When you specify a character string for the pushbutton with *text* and an image with *disabledBitmap*, the image is displayed with the grayed-out character string

disablePopup

Topic group Related topics

Whether the tree view's popup menu is disabled.

Property of

TreeView

Description

The tree has a right-click popup menu that allows the user to insert, delete, and edit items. Set *disablePopup* to *true* to disable this popup menu.

Even if the popup is disabled, the user can still edit the items by clicking twice or pressing F2, and insert and delete items by pressing Ins and Del. Set *allowEditLabels* and *allowEditTree* to *false* to prevent these actions.

doVerb( )

Topic group Related topics Example

Starts an action in an OLE server application.

Syntax

<oRef>.*doVerb*(<verb expN> [, <title expC>])

<oRef>

The OLE control that contains the linked or embedded object.

<verb expN>

The numeric value of the OLE verb.

<title expC>

An optional text string to display in the title bar of the server window.

Property of

OLE

Description

Use *doVerb*( ) to initiate an action from an OLE document stored in an OLE field and to specify what action to take.

Every OLE object accepts one or more verbs. Each verb determines which actions are taken, and each is represented by a number.

downBitmap

Topic group Related topics

Specifies the graphic image to display in a pushbutton when the user presses the button.

Property of

PushButton

Description

Use *downBitmap* to give visual confirmation when the user clicks a pushbutton. When the user releases the mouse button or moves the pointer off the pushbutton, the image and/or text specified by *focusBitmap* and *text* is displayed.

The *downBitmap* setting can take one of two forms:

RESOURCE <resource id> <dll name>
specifies a bitmap resource and the DLL file that holds it.

FILENAME <filename>
specifies a bitmap file. See class Image for a list of bitmap formats supported by *dBASE Plus*.

When you specify a character string for the pushbutton with *text* and an image with *downBitmap*, the image is displayed with the character string.

drag( )

Topic group Related topics Example

Initiates a Drag&Drop Copy or Move operation for a *dBASE Plus* UI object.

Syntax

<oRef>.*drag*(<type expc>, <name expC>, <icon expC>)

<oRef>

The object to be copied or moved.

<type expc>

A string, typically identifying the object's type.

<name expC>

A string, typically containing the name of the object.

<icon expC>

The filename of a cursor icon to be displayed while the object is being dragged. This parameter is required, but is currently unused. The default Windows OLE cursor will be displayed.

Property of

Many Form objects

Description

Use *drag*( ) to initiate a Drag&Drop operation for a Drop Source object. Drop Source objects may only be dropped upon “active” Drop Target objects; that is, any object whose *allowDrop* property is set to true. The *drag*( ) method is typically called from within the Drop Source’s *onLeftMouseDown* event handler.

drag( ) returns true or false, according to the success of the drop operation.

For Copy operations, <type expC> and <name expC> are passed directly from a Drop Source’s *drag*( ) method to a Drop Target’s *onDragEnter*, *onDragOver*, and *onDrop* events. Other than a length restriction of 260 characters, these parameters have no mandatory format and may be used to communicate any information.

The type of operation initiated is determined by the object’s *dragEffect* property, and the state of the Control key when the mouse button is pressed. The following table shows the possible settings and resulting operations:

dragEffect	Control key	Operation type
0 - None	(ignored)	None (dragging disabled)

1 - Copy	(ignored)	Copy
2 - Move	up	Move
2 - Move	down	Copy

For a Move operation, the dragged object moves with the mouse and may only be dropped within its containing object, e.g. a parent Form or Container. The operation will only take place if the containing object is also an active Drop Target. The Move operation terminates when the mouse button is released. The Target's *onDrop* event is not fired for Move operations.

For a Copy operation, the dragged object appears to remain in place and may be dropped either within its containing object, or on any active Drop Target object. When the mouse button is released, the Drop Target's *onDrop* fires, processes the event, then returns true or false to the initiating *drag*( ) method.

dragEffect

Topic group Related topics

Specifies a Drag&Drop processing mode for a Drop Source object

Property of

Many Form objects

Description

Use *dragEffect* to enable or disable dragging for a Drop Source object, and to specify the default drag processing mode.

Each time a Drag&Drop operation is attempted for a Drop Source object, the value of *dragEffect* and the state of the Control key are evaluated. The following table shows the possible values for *dragEffect*, and the effect of the Control key on the resulting operation:

dragEffect	**Control key**	**Operation type**
0 - None (default)	N/A	None (dragging disabled)
1 - Copy	(ignored)	Copy
2 - Move	up	Move
2 - Move	down	Copy

An object becomes an active Drop Source when the value of its *dragEffect* property is set greater than 0. Similarly, an object becomes an active Drop Target when its *allowDrop* property is set to true. Except for forms, all Drop Target objects may also be Drop Sources.

dragScrollRate

Topic group Related topics

The delay time between each column scroll when dragging columns.

Property of

Grid

Description

The *dragScrollRate* property controls how fast the grid scrolls horizontally when rearranging columns in the grid by dragging. The delay time is measured in milliseconds.

drawMode

Topic group Related topics

Specifies how the colors of a Shape object appear on the underlying surface.

Property of

Shape

Description

Use *drawMode* to create visual effects with the pen and brush (fill) colors of a Shape object.

dBASE Plus compares the colors specified by the *colorNormal* property of the Shape to the color of the underlying surface (usually a form), and renders a result according to *drawMode*.

In the table below, "Pen" refers to both the pen and brush colors of the Shape. "Merge" (bitwise OR), "Mask" (bitwise AND), and "XOR" indicate the type of bitwise operation to be performed on the pixel RGB values of the "Pen" and the drawing surface, while "NOT" means the operation involves the inverse, or complement (color negative), of one or both colors.

You can specify any of these settings for *drawMode*:

Value	Description
0 – Normal	Color specified in the Shape *colorNormal* property (100% opacity)
1 – Inverse	Inverse (color negative) of the drawing surface color. **Note:** this setting *disregards* the Pen color.
2 – Merge Pen NOT	Combination of the Pen color and the *inverse* of the drawing surface color
3 – Mask Pen NOT	Combination of the colors common to *both* the Pen and the *inverse* of the surface
4 – Merge NOT Pen	Combination of the surface color and the *inverse* of the Pen color
5 – Mask NOT Pen	Combination of the colors common to *both* the surface and the *inverse* of the Pen
6 – Merge Pen	Combination of the Pen color and the surface color
7 – NOT Merge Pen	Inverse (color negative) of Merge Pen
8 – Mask Pen	Combination of the colors common to *both* the Pen and the surface
9 – NOT Mask Pen	Inverse (color negative) of Mask Pen
10 – XOR Pen	Combination of the colors in the Pen and the surface, but *not* common to both
11 – NOT XOR Pen	Inverse (color negative) of XOR Pen

dropDownHeight

Topic group Related topics

The number of lines displayed in the list portion of the combobox.

Property of

ColumnEditor, ComboBox

Description

Use *dropDownHeight* to specify how much information will appear when a user drops down a list from a combo box.

dropDownWidth

Topic group Related topics

The width of the list portion of the combobox.

Property of

ComboBox

Description

The width of the drop-down list of a combobox may be different than the width of the text entry portion. If *dropDownWidth* is zero (the default), the width of the drop-down list is sized to match the width of the control. Otherwise, the combobox's *dropDownWidth* setting is used.

The *dropDownWidth* is expressed in the form's current *metric* units (the same as the *width*).

editorControl

Topic group Related topics

The editable control that comprises the body of the grid in the column.

Property of

GridColumn

Description

The *editorControl* property contains an object reference to the editable control in the grid column. The type of control is determined by the *editorType* property.

editorType

Topic group Related topics

The type of editing control in the grid column.

Property of

GridColumn

Description

The *editorType* property is an enumerated property that determines the type of editable control used for the data in the column. It may have one of the following values:

Value	Description	Control
0	Default	Depends on column data type
1	Entryfield	ColumnEntryfield
2	CheckBox	ColumnCheckBox
3	SpinBox	ColumnSpinBox
4	ComboBox	ColumnComboBox

5	Editor	ColumnEditor

You may access the control through the *editorControl* property.

elements

Topic group Related topics Example

An array containing object references to all the components in a form.

Property of

Form, SubForm

Description

The *elements* array contains an object reference for each visual component in a form. Other types of objects, like data objects, are not included.

You can determine the number of components in the form by checking the *elements* array's *size* property. Each element in the array can be addressed by its element number or by the *name* of the component.

The *elements* array is not a member of the Array class, but rather an ObjectArray class with specific capabilities for managing a list of objects. It does not support most of the methods of the Array class. The *elements* array is not meant to be changed directly, although it is safe to scan to get the object references for the components in the form.

enabled

Topic group Related topics Example

Determines if an object can get focus and operate.

Property of

Most form components

Description

When you set the *enabled* property of an object to *true*, the user can select and use the object. When you set the *enabled* property to *false*, the object is dimmed and the user can't select or use the object. This is a visual indication that the object cannot get focus. The object is removed from the tab sequence and clicking the object has no effect.

The *enabled* properties of the Container and Notebook components differ somewhat from those of other form components. When the Container or Notebook's *enabled* property is set to "false", the enabled properties of all contained controls are likewise set to "false". When it's set to "true", the enabled properties of the contained controls regain their individual settings.

For a form, setting enabled to false will disable all components on a form or subform. The form's border area controls (the non-client area) will remain functional.

enableSelection

Topic group Related topics

Whether the selection range is displayed in the Slider object.

Property of

Slider

Description

If *enableSelection* is *true*, the selection range set by the slider's *startSelection* and *endSelection* properties is displayed inside the slider as a colored area with matching tic marks. You may use the selection range to show the current or preferred range.

If *enableSelection* is *false*, the *startSelection* and *endSelection* properties have no effect.

endSelection

Topic group Related topics

The high end of the selection range in a Slider object.

Property of

Slider

Description

endSelection contains the high value in the selection range. It should be equal to or higher than *startSelection*, and between the *rangeMin* and *rangeMax* values of the slider.

The selection is not displayed unless the slider's *enableSelection* property is *true*.

ensureVisible( )

Topic group Related topics

Makes the tree item visible in the tree view.

Syntax

<oRef>.*ensureVisible*( )

<oRef>

An object reference to the tree item you want to display.

Property of

TreeItem

Description

Use *ensureVisible*( ) when you want to make sure that a tree item is visible in the tree view. The tree is expanded and scrolled if necessary to make the item visible. If the tree item is already visible, nothing happens.

escExit

Topic group Related topics

Determines if the user can close a form by pressing Esc.

Property of

Form, SubForm

Description

Set *escExit* to *false* to prevent the user from closing a form by pressing Esc.

You can verify that the user wants to close the form by using the form's *canClose* event handler. Closing a form by pressing Esc abandons all pending changes in the data buffer, as if *abandonRecord*( ) was called.

When a form is opened with *readModal*( ), it returns *false* when it is closed by pressing Esc.

evalTags

Topic group Related topics

Whether to evaluate HTML tags in the text.

Property of

ColumnEditor, Editor

Description

dBASE Plus supports common HTML formatting tags. You may choose to evaluate any tags that appear in the text of the editor and apply the formatting, or to leave the HTML tag as-is so that they can be edited.

Use the Format toolbar to format the text with HTML tags. You may also type in the tags manually, but they will not be interpreted until you toggle *evalTags* to *true*.

The editor's *evalTags* property corresponds to the Apply Formatting option in the editor's popup menu. The Format toolbar is not active if the editor's *evalTags* property is *false*.

expanded

Topic group Related topics

Whether the tree item's children are shown or hidden.

Property of

TreeItem

Description

A tree item's subtree may be expanded or collapsed visually by the user, or by assigning a value to the *expanded* property. If *expanded* is *true*, the subtree is displayed. If *expanded* is *false*, the subtree is hidden.

fields

Topic group Related topics Example

Specifies the fields to display in a Browse object, and the field options to apply to each field.

Property of

Browse

Description

The *fields* is a string with the following the following format:

<field 1> [<field option list 1>] |
<calculated field 1> = <exp1> [<calculated field option list 1>]
  [, <field 2> [<field option list 2>] |
  <calculated field 2> = <exp1> [<calculated field option list 2>], ...]

The fields are displayed in the order they're listed, and the <field option list> affects the way each field is displayed. The options are as follows:

Option	Description
\<column width>	The width of the column within which the field appears
\B = <exp 1>, <exp 2>	RANGE option; forces any value entered in <field 1> to fall within <exp 1> and <exp 2>, inclusive.
\H = <expC>	HEADER option; causes <expC> to appear above the field column in the browse, replacing the field name
\P = <expC>	PICTURE option; formats the field according to the *picture* template <expC>
\V = <condition> [\E = <expC>]	VALID option; allows a new field value to be entered only when <condition> evaluates to *true* ERROR MESSAGE option; \E = <expC> causes <expC> to appear when <condition> evaluates to *false*

Calculated fields are read-only and are composed of an assigned field name and an expression that results in the calculated field value, for example:

```
Commission = Rate * Saleprice
```

Options for calculated fields affect the way these fields are displayed. These options are as follows:

Option	Description
\<column width>	The width of the column within which the calculated field is displayed
\H = <expC>	Causes <expC> to appear above the calculated field column in the browse, replacing the calculated field name

filename

Topic group Related topics

The name of the file that contains the desired report.

Property of

ReportViewer

Description

Set the *filename* property to the file that contains the report class definition. Reports are stored in files with a .REP extension.

The .REP file is executed when you assign the *filename* property, even if the form is not open. Normally, report parameters (if any) are assigned to the *params* array before setting the *filename* property; if they are assigned after setting the *filename* property, you must call the ReportViewer object's *reExecute*( ) method to regenerate the report.

first

Topic group Related topics Example

A reference to the first component in a containing object's z-order.

Property of

Form, Container, Notebook, SubForm

Description

Use the *first* property to reference the first component in the containing object's z-order. For more information on component z-order within containing objects, see *before*.

In forms, if the *first* component can receive focus, it gets focus initially when you open the form. Otherwise, the next component in the z-order is tried until one is found that can receive focus.

first is a read-only property.

firstChild

Topic group Related topics Example

The first child tree item.

Property of

TreeItem, TreeView

Description

firstChild is a read-only property that contains an object reference to the object's first child tree item. If the object has no children, *firstChild* is *null*.

When a form is opened, a TreeView object's *firstChild* property will point to it's first TreeItem. The *firstChild* property will return "null" when:

The form's TreeView doesn't contain any TreeItems

The form isn't open.

firstColumn

Topic group Related topics

Index of the column to be displayed in the left-most unlocked column position.

Property of

Grid

Description

The *firstColumn* property is an index, starting with the left-most unlocked column equal to one, that specifies the first column to be displayed after the locked columns. Setting this property will cause the selected column to be scrolled to the left-most unlocked column display position. While the *firstColumn* property is a writable, its value is not streamed to the source code by the form designer, so when the form containing the grid is opened during run mode, the *firstColumn* property is initialized to its default value of one.

As implied above, the *lockedColumns* property affects the display position and the index values associated with the *firstColumn* property. To avoid unexpected results, these properties are

implemented so any change in the *lockedColumns* property will reset the *firstColumn* property to one.

firstRow( )

Topic group Related topics Example

Returns a bookmark for the row currently displayed in the first row of the grid.

Syntax

<oRef>.*firstRow*( )

<oRef>

A reference to a grid object.

Property of

Grid

Description

Calling the *firstRow*( ) method returns a bookmark to the row currently displayed on the grid's first, or top, row. If the grid is not datalinked to any rowset, the *firstRow*( ) method returns null.

The *firstRow*( ) method can be used to preserve, and later return to, the top displayed row in the grid. Use a rowset's *goto*( ) method to position a rowset to a previously bookmarked row (see example).

firstVisibleChild

Topic group Related topics

The first tree item that is visible in the tree view area.

Property of

TreeView

Description

firstVisibleChild is a read-only property that contains an object reference to the first tree item that is visible at the top of the tree view area. Note that this is not necessarily a top-level tree item.

focus

Topic group Related topics

When to give focus to the notebook tabs when they are clicked.

Property of

NoteBook

Description

The *focus* property determines whether the notebook tabs (or buttons if *buttons* is *true*) get focus when they are clicked. It is an enumerated property with the following possible values:

Setting	Description
0 (Normal)	Tab does not focus if tab changes
1 (On Button Down)	Tab always gets focus
2 (Never)	Tab never gets focus

You can always give focus to the notebook tabs with the Tab and Shift+Tab keys if its *tabStop* property is *true*. If a tab already has focus, clicking will keep focus in the tabs, no matter what the *focus* property is.

focusBitmap

Topic group Related topics

Specifies the graphic image to display in a pushbutton when the pushbutton has focus.

Property of

PushButton

Description

Use *focusBitmap* to indicate visually when a pushbutton is selected.

The *focusBitmap* setting can take one of two forms:

RESOURCE <resource id> <dll name>
specifies a bitmap resource and the DLL file that holds it.

FILENAME <filename>
specifies a bitmap file. See class Image for a list of bitmap formats supported by *dBASE Plus*.

When you specify a character string for the pushbutton with *text* and an image with *focusBitmap*, the image is displayed with the character string.

fontBold

Topic group Related topics

Specifies whether the component displays its text in bold type.

Property of

Many form components

Description

Set *fontBold* to *true* if you want the component to display its text in boldface.

For Grid objects, the *fontBold* property can be overridden by setting the *fontBold* property of a GridColumn's *editorControl* to a non-null value.

fontItalic

Topic group Related topics

Specifies whether the component displays its text in italics.

Property of

Many form components

Description

Set *fontItalic* to *true* if you want the component to display its text in italics.

For Grid objects, the *fontItalic* property can be overridden by setting the *fontItalic* property of a GridColumn's *editorControl* to a non-null value.

fontName

Topic group Related topics

The typeface of the component's text.

Property of

Many form components

Description

Set the *fontName* property to the name of the typeface you want to apply to the text in the component. For Grid objects, the *fontName* property can be overridden by setting the *fontName* property of a GridColumn's *editorControl* to a non-null value. The *fontName* property defaults to that set by your operating system or the PLUS.ini file.

Note:
In order to use TrueType fonts in European countries which do not use the Western Europe code page (1252), you must specify the language (also referred to as the "script").

Since *dBASE Plus* does not list available language scripts for TrueType fonts, you must specify it in the fontName property--either in code or through the Inspector--using the exact TrueType font name. To do it through the Inspector, for example, choose a text component, choose the fontName property in the Inspector, and, instead of choosing from available fonts on the list, type in the name of the language script. We recommend all languages be entered in English, e.g.:

```
Times New Roman Greek
Verdana Turkish
Arial Baltic
MS Gothic Cyrillic
Courier New Central Europe
```

The following PLUS.ini file settings ensure that the initial font created for a new control uses the language you want:

```
[DefaultFonts]
Application=<strFontName>,<intPointSize>
Controls=<strFontName>,<intPointSize>
```

The Application setting specifies the font used for the Navigator and Inspector, while the Controls setting specifies the default font used for forms and controls. You can also create your own custom controls to specify the font and language you want to use.

fontSize

Topic group Related topics

The point size of the component's text.

Property of

Many form components

Description

Set *fontSize* to the point size in which you want the text of the component to be displayed. There are approximately 72 points per inch. Common text is from 8 to 12 points. Default: 10.

For Grid objects, the *fontSize* property can be overridden by setting the *fontSize* property of a GridColumn's *editorControl* to a non-null value.

fontStrikeout

Topic group Related topics

Specifies whether the component displays its text struck through.

Property of

Many form components

Description

Set *fontStrikeout* to *true* if you want the component to display its text struck through.

For Grid objects, the *fontStrikeout* property can be overridden by setting the *fontStrikeout* property of a GridColumn's *editorControl* to a non-null value.

fontUnderline

Topic group Related topics

Specifies whether the component displays its text underlined.

Property of

Many form components

Description

Set *fontUnderline* to *true* if you want the component to display its text underlined.

For Grid objects, the *fontUnderline* property can be overridden by setting the *fontUnderline* property of a GridColumn's *editorControl* to a non-null value.

form

Topic group Related topics Example

The form or report that contains the component.

Property of

All form components.

Description

A component's *form* property is a reference to the form or report that contains it. It is set automatically when the component is created and cannot be changed.

Use the *form* property in component event handlers and methods to generically refer to the object that contains the component.

In a form, a component's *form* and *parent* property refer to the same thing—the form—if the component is placed directly on the form. However, if for example you place a component in a NoteBook control on a form, then the component's *parent* is the NoteBook control, not the form; and the NoteBook control's *parent* is the form. In a report, components are contained deeper in the object hierarchy and their *parent*s are never the report.

By using the *form* property, you can immediately get back to the top of the object hierarchy and refer to its properties, events, or methods; or refer to other objects in the form or report.

In addition to the *form* property, a local variable named *form* is also created for any event handler for form, report, or a visual component. The *form* variable is not created for events in other kinds of classes. The variable and the *form* property both refer to the same object. The *form* variable is used in most event handlers.

frozenColumn

Topic group Related topics

The field name of the column in which the cursor is confined.

Property of

Browse, Grid

Description

Use the *frozenColumn* property to confine the cursor in a single column of the grid. The other columns in the grid are displayed, but you cannot put the cursor in them.

The *frozenColumn* property expects the field name of the column (a string). To release the cursor from the column, assign an empty string to the *frozenColumn* property.

function

Topic group Related topics Example

Formats text in an object.

Property of

Entryfield, SpinBox, Text

Description

Use a formatting *function* to format the display and entry of data. While a *picture* gives you character-by-character control, a *function* formats the entire value.

dBASE Plus recognizes the following *function* symbols:

Symbol	Description
(	Encloses negative numbers in parentheses.
!	Converts letters to uppercase.
#	Restricts entries to numbers, signs, spaces and the SET POINT character.
^	Displays numbers in exponential form.
$	Inserts a dollar sign or the symbol defined with SET CURRENCY TO instead of leading spaces.
A	Restricts entry to alphabetic characters.

B	Left-aligns a numeric entry.
C	Displays CR (credit) after a positive number.
D	Displays and accepts entry of a date in the current SET DATE format. **D** should be the last function code when used with other function codes.
E	Displays and accepts entry of a date in European (DD/MD/YY) format. **E** should be the last function code when used with other funciton codes.
I	Centers the entry.
J	Right-aligns the entry.
L	Displays numbers with leading zeros. Overrides the **$** function code and the **$** and ***** picture codes.
R	Inserts literal placeholders, characters that are not template codes, into the display without storing them in the field. Literal placeholders, included after the **R** function code in a picture template, will be displayed with the value but will not be stored as data for the object. For example, if the value of a field named "Phone" is 4155551212, a template of "@R (999) 999-9999" displays the phone number as (415) 555-1212 but stores it as 4155551212. **R** must be the last function code if it is used with other function codes.
S<n>	Truncates a character string to a width specified by **<n>**.
T	Removes leading and trailing spaces from character values.
V<n>	? and ?? command only: Wraps a character string within a width specified by **<n>**.
X	Displays DB (debit) after a negative number.
Z	Displays zeros as a blanks.

getColumnObject( )

Topic group Related topics

Returns a reference to the GridColumn object for a designated column

Syntax

<oRef>.*getColumnObject*(<exp N>)

<oRef>

The name of the Grid object

<exp N>

An integer representing the column position. For the leftmost column in a grid, n=1.

Property of

Grid

Description

Each column in a grid is represented by a GridColumn object. For a grid that has custom columns defined, the *getColumnObject*( ) method will return a reference to the GridColumn object for column position <n>.

<exp n> is an integer from 1 up to the number of columns in the grid where;

n=1 indicates the current leftmost column

n=2 indicates the current second column from the left

and so on.

The *getColumnObject*( ) method provides a means to determine the current column order, save the column order to disk and restore it later on.

If a grid does NOT have custom columns defined, the *getColumnObject*( ) method returns a null value.

getColumnOrder( )

Topic group Related topics Example

Returns a two-dimensional array, the columns of which are *QueryName* and *FieldName.*

Syntax

<oRef>.*getColumnOrder*( )

<oRef>

The name of the Grid object

Property of

Grid

Description

When the *allowColumnMoving* property is set to it's default setting, *true*, the user is able to rearrange columns in a grid by clicking and dragging the column headings. Using the *getColumnOrder*( ) method, the new array values can be saved using the form's *onClose* event and subsequently restored using the form's *onOpen* event.

getItemByPos( )

Topic group Related topics Example

Returns an object reference to a TreeItem object located at a specified position.

Syntax

[<oRef> =] *getItemByPos*(<col expN>, <row expN>)

<oRef>

A variable or property in which to store the TreeItem object reference returned by *getItemByPos*( ).

<col expN>

The horizontal position, within a TreeView object, to check for a TreeItem.

<row expN>

The vertical position, within a TreeView object, to check for a TreeItem.

Property of

TreeView

Description

Use *getItemByPos*( ), within a TreeView mouse event handler, to determine on which TreeItem object the user clicked. If no TreeItem was clicked, *getItemByPos*( ) returns a null value.

getTextExtent( )

Topic group Related topics

Returns the length of a text string based on the current font settings of the Text or TextLabel control.

Syntax

<oRef>.*getTextExtent*(<expC>)

<oRef>

The Text or TextLabel object used to calculate the text size.

<expC>

The string to measure

Property of

Text, TextLabel

Description

The *getTextExtent*( ) method calculates the *width* required to display <expC>, in the Text or TextLabel object, using the object's current font settings. It returns a value in the form's current *metric*.

the *wrap* property

When a Text object's *wrap* property is *false*, the value returned by the *getTextExtent* property will equal the length of the line of text. However, when the *wrap* property is set to *true*, the value returned by the *getTextExtent* property will only reflect the length of the text before a wrap occurs.

gridLineWidth

Topic group Related topics

The width of the grid lines in a Grid object.

Property of

Grid

Description

gridLineWidth controls the width, in pixels, of the lines that separate the cells in a Grid object.

group

Topic group Related topics Example

Creates component groups in the form's tab order.

Property of

CheckBox, PushButton, RadioButton

Description

Use *group* to determine if an object is part of a group within which the user can move focus with the arrow keys. Pressing Tab does not move the focus within the group; it moves to the next object outside the group. All objects in a group must be of the same class.

RadioButtons must be used in groups of two or more. Only one RadioButton in the group may be selected at any time.

You may also use *true* and *false* to create groups. Set the *group* property of the first object in the group to *true*. For all other objects that belong to the group, set *group* to *false*. The next object with a *group* setting of *true* begins another group.

handle [Form objects]

Topic group Related topics

The Windows tree item handle

Property of

TreeItem

Description

Each tree item has an internal handle. This handle is similar to the *hWnd* handle for each control on a form. This handle may be used for low-level API calls to manipulate the individual items in the tree view

hasButtons

Topic group Related topics

Whether + and - icons are displayed for tree items that have children.

Property of

TreeView

Description

When *hasButtons* is *true*, tree items that have child items have a + or - icon to indicate whether the subtree is collapsed or expanded. Clicking the icon expands and collapses the tree.

Set *hasButtons* to *false* to prevent these icons from appearing. The user can still expand and collapse a subtree by pressing Shift+minus and Shift+plus on the numeric keypad. To prevent these keys from expanding or collapsing the tree, use the *canExpand* event.

hasColumnHeadings

Topic group Related topics

Whether column headings are displayed.

Property of

Grid

Description

Set *hasColumnHeadings* to *false* to suppress column headings in a grid. If *hasIndicator* is also *false*, the grid will contain data cells only. *hasColumnHeadings* must be set to *true* to allow the user to move, or resize, columns in a grid.

hasColumnLines

Topic group Related topics

Whether column (vertical) grid lines are displayed.

Property of

Grid

Description

Set *hasColumnLines* to *false* to suppress the vertical lines that separate columns in the grid.

hasIndicator

Topic group Related topics

Whether the indicator column is displayed.

Property of

Grid

Description

The indicator column is the left-most column in the grid, and contains an icon indicating the current column. The icon changes when a row is being appended.

Set *hasIndicator* to *false* to suppress the indicator column. If *hasColumnHeadings* is also *false*, the grid will contain data cells only.

hasLines

Topic group Related topics

Whether lines are drawn between tree items.

Property of

TreeView

Description

Set *hasLines* to *false* to display items in the tree view without the normal connecting branch lines. To disable lines at the root level only, leave *hasLines true* and set *linesAtRoot* to *false*.

hasRowLines

Topic group Related topics

Whether row (horizontal) grid lines are displayed.

Property of

Grid

Description

Set *hasRowLines* to *false* to suppress the horizontal lines that separate rows in the grid.

hasVScrollHintText

Topic group Related topics

Whether the relative row count is displayed as the grid is scrolled vertically.

Property of

Grid

Description

When *hasVScrollHintText* is *true*, a relative row count, like "12 of 600" is continuously updated and displayed next to the vertical scrollbar as it is scrolled.

Set *hasVScrollHintText* to *false* to suppress the message.

headingColorNormal

Topic group Related topics

Determines the text and background color for grid column heading controls.

Property of

Grid

Description

Use the *headingColorNormal* property to set the text color and background color for grid column heading controls. This property can be overridden by setting a grid column headingControl's *colorNormal* property to a valid non-null value. The default for the *headingColorNormal* property is WindowText/BtnFace.

headingControl

Topic group Related topics

The control that displays the grid column heading.

Property of

GridColumn

Description

The *headingControl* property contains an object reference to the ColumnHeadingControl object that contains the column heading. The editable control in the column is referenced through the *editorControl* property.

headingFontBold

Topic group Related topics

Determines whether the current heading font style is Bold.

Property of

Grid

Description

When the *headingfontBold* property is set to *true*, sets the current heading font style to bold. This property can be overridden by setting a grid column headingControl's *fontBold* property to a non-null value. The *headingFontBold* property defaults to *true.*

headingFontItalic

Topic group Related topics

Determines whether the current heading font style is Italic.

Property of

Grid

Description

When the *headingfontItalic* property is set to *true*, sets the current heading font style to italic. This property can be overridden by setting a grid column headingControl's *fontItalic* property to a non-null value. The *headingFontItalic* property defaults to *false.*

headingFontName

Topic group Related topics

Determines to font used to display data in a grid's headingControls

Property of

Grid

Description

Use the *headingFontName* property to set the font used to display data in a grid's headingControls. The *headingFontName* property can be overridden by setting a grid column headingControl's *fontName* property to a valid non-null value.

The *headingFontName* property defaults to that set by your operating system, or the PLUS.ini file.

headingFontSize

Topic group Related topics

Determines the character size of the current heading font.

Property of

Grid

Description

When the *headingfontSize* property is set to *true*, sets the font's character size for data displayed in a grid's headingControls. This property can be overridden by setting a grid column headingControl's *fontSize* property to a value greater than zero. The *headingFontSize* property defaults to 10 points.

headingFontStrikeout

Topic group Related topics

Determines whether the current heading font style is Strikeout.

Property of

Grid

Description

When the *headingFontStrikeout* property is set to *true*, displays the current heading font with a horizontal strikeout line through the middle of each character. The *headingFontStrikeout* property can be overridden by setting a grid column headingControl's *fontStrikeout* property to a non-null value. The *headingFontStrikeout* property defaults to *false.*

headingFontUnderline

Topic group Related topics

Determines whether the current heading font style is Underline.

Property of

Grid

Description

When the *headingFontUnderline* property is set to *true*, the current heading font style is set to Underline. This property can be overridden by setting a grid column headingControl's *fontUnderline* property to a non-null value. The *headingFontUnderline* property defaults to *false.*

height

Topic group Related topics

The height of an object. For Form and SubForm objects, the height of their client areas.

Property of

Form, SubForm, Form objects and Report objects: Band, PageTemplate, StreamFrame.

Description

Form objects: The value of the *height* property includes any border, bevel or shadow effect assigned to the object.

Forms and SubForms: The value of the *height* property includes only the client area. It does not include the window border or titlebar.

When a Form or SubForm is opened, and has a horizontal scrollbar (see the *scrollbar* property), the Form or SubForms' width is automatically reduced by the width of the horizontal scroll bar (16 pixels).

The minimum height of a Form or SubForm is zero.

The *height* property is numeric and expressed in the current *metric* unit of the Form or Subform that contains the object.

The unit of measurement in a form or report is controlled by its *metric* property. The default metric for forms is "characters", and for reports it’s "twips".

helpFile

Topic group Related topics

Identifies a help file that contains context-sensitive Help topics. This can be set to either a Windows help file (.hlp) or a Microsoft Compiled Html Help file (.chm).

Property of

Most form objects.

Description

Use *helpFile* in combination with *helpId* to provide context-sensitive help from a Windows Help file. Context-sensitive help appears for the object that has focus when the user presses F1, or chooses Help | Context Sensitive Help from the default menu.

After creating the Windows Help file, follow these steps:

1. Assign the name of the Help file to the form's *helpFile* property. This assigns the default Help file for all help topics.
2. Assign a context ID or index string for the form's default Help topic to the form's *helpId* property.
3. For individual controls that have their own help, assign the appropriate value to the control's *helpId* property.
4. If an individual control has a topic in a different Help file, assign that file to the control's *helpFile* property.

Warning!

If you assign the F1 key as the *shortCut* key to your own menu item, pressing F1 executes the *onClick* for that menu item; it does not display context-sensitive help. Context-sensitive help is also disabled if you assign an *onHelp* event handler.

helpId

Topic group Related topics

Specifies the Help context ID or index entry for an object.

Property of

Most form objects

Description

Use *helpId* in combination with *helpFile* to assign context-sensitive help to a control. Context-sensitive help appears for the object that has focus when the user presses F1, or chooses Help | Context Sensitive Help from the default menu.

Warning!

If you assign the F1 key as the *shortCut* key to your own menu item, pressing F1 executes the *onClick* for that menu item; it does not display context-sensitive help. Context-sensitive help is also disabled if you assign an *onHelp* event handler.

The *helpId* is a string that contains either:

A context ID number, preceded by the "#" symbol, for example:

```
#2002
```

Choosing help displays the topic with that context ID number. If that context ID is not found, Help displays an error.

A help index string, for example

```
Deleting accounts
```

Choosing help searches the index for that string. If only one topic is found that uses that index string, it is displayed. If there are multiple matches, Help displays a Topics Found dialog, letting the user choose which topic to view. If the string is not found in the index, the Help index is displayed, with the *helpId* property as the current search value.

As with *helpFile*, you may set a default *helpId* in the form. If the control that has focus does not have its own *helpId* property, the form's value is used.

hScrollBar

Topic group Related topics

Determines when an object has a horizontal scroll bar.

Property of

Grid

Description

The *hScrollBar* property determines when and if a control displays a horizontal scrollbar. It may have any of four settings:

Value	Description
0 (Off)	The object never has a horizontal scroll bar.
1 (On)	The object always has a horizontal scroll bar.
2 (Auto)	Displays a horizontal scroll bar only when needed.
3 (Disabled)	The horizontal scroll bar is visible but not usable.

hWnd

Topic group Related topics

The Windows object handle for the form object.

Property of

Most form objects

Description

Use the *hWnd* property when you need to pass the handle of a form object to a Windows API function or other external DLL.

hWnd vs hWndClient

hWndClient is the handle for the parent window that contains a form's controls. In contrast, *hWnd* is the handle for the form itself; the parent of the *hWndClient* window, and the grandparent of the controls.

The *hWnd* property is read-only.

hWndClient

Topic group Related topics

The Windows object handle of the window that contains a form's controls.

Property of

Form, SubForm

Description

hWndClient is the handle for the parent window that contains a form's controls. In contrast, *hWnd* is the handle for the form itself; the parent of the *hWndClient* window, and the grandparent of the controls.

hWndParent

Topic group Related topics Example

For a non-mdi form (a form with form.mdi = false), the *hWndParent* property can be used in conjunction with the *showTaskBarButton* property to determine, or specify, the *hWnd* property for the parent window of a form.

Property of

Form

Description

When a form's *showTaskBarButton* property is set to *false*, calling the form's *open*( ) method will cause its *hWndParent* property to be set to the *hWnd* property of a hidden parent window.

Windows will not create a Taskbar button for a window if it has a parent window associated with it.

Alternatively, before opening a non-mdi form, you can set the *hWndParent* property to an open form's *hWnd* property. If you also set the *showTaskBarButton* property to *false*, the specified *hWndParent* will be assigned as the parent window for the non-mdi form when that form is opened (*see example*).

Switching between dBASE Plus, or a dBASE Application, and other Windows programs

When running a non-modal and non-mdi form, you must set the form's *hWndParent* property to the appropriate parent hWnd (ex. _app.frameWin.hWnd), during the form's *open*( ) method, to ensure the form will always stay on top when switching between dBASE Plus, or a dBASE Application, and other Windows programs.

icon

Topic group Related topics

Specifies an icon file (.ICO) or resource that displays when a form is minimized.

Property of

Form, SubForm

Description

Use *icon* to specify an image to be used when a form is minimized. The *icon* property is a string that can take one of two forms:

RESOURCE <resource id> <dll name>
specifies a bitmap resource and the DLL file that holds it.

FILENAME <filename>
specifies an .ICO file.

id

Topic group Related topics

Identifies an object with a numeric value.

Property of

Most form components

Description

Use *id* to give a unique supplementary identifier to an object.

In most cases, you use an object's *name* or compare object references directly to determine the identity of an object. *id* is used primary with the *onSelection* event, which is an antiquated and rarely-used event.

The *id* property defaults to -1.

image

Topic group Related topics

Image displayed between checkbox and text label when a tree item does not have focus.

Property of

TreeItem, TreeView

Description

The tree view may display images to the left of the text label of each tree item. If the tree has checkboxes, the image is displayed between the checkbox and the text label.

The *image* property of the TreeView object specifies the default icon image for all tree items when they do not have focus. You may designate specific icons for each TreeItem object to override the default. Use the *selectedImage* property to specify icons for when a tree item has focus. If any individual item in the tree has its *image* or *selectedImage* property set, space is left in all tree items for an icon, even if they don't have one.

The *image* property is a string that can take one of two forms:

RESOURCE <resource id> <dll name>
specifies an icon resource and the DLL file that holds it.

FILENAME <filename>
specifies an ICO icon file.

imageScaleToFont

Topic group Related topics

Whether tree item images automatically scale to match the text label font height.

Property of

TreeView

Description

When *imageScaleToFont* is *true*, the *image*, *selectedImage*, *checkedImage*, *uncheckedImage*, and default checkbox images all scale to match the height of the text label font, controlled by the *fontName* and *fontSize* properties.

imageSize

Topic group Related topics

The height of tree item images in pixels.

Property of

TreeView

Description

imageSize reflects the height of the *image*, *selectedImage*, *checkedImage*, *uncheckedImage*, and default checkbox images in a tree view. You may assign a size if *imageScaleToFont* is *false*.

imgPixelHeight

Topic group Related topics

Returns an image's actual height in pixels.

Syntax

<oRef>.*imgPixelHeight*

<oRef>

A reference to an image object.

Property of

Image

Description

The *imgPixelHeight* property is readonly and returns the actual height of the image currently loaded into an image object.

When no image is loaded into an image object, the *imgPixelHeight* property returns 0 (the default).

The *imgPixelHeight* property can be used with the *imgPixelWidth* property to retrieve the actual size of an image in pixels.

The image object's *height* and *width* properties can then be set to display the image at its actual size without clipping part of the image, or scaling the image to fit the image object.

imgPixelWidth

Topic group Related topics

Returns an image's actual width in pixels.

Syntax

<oRef>.*imgPixelWidth*

<oRef>

A reference to an image object.

Property of

Image

Description

The *imgPixelWidth* property is readonly and returns the actual width of the image currently loaded into an image object.

When no image is loaded into an image object, the *imgPixelWidth* property returns 0 (the default).

The *imgPixelWidth* property can be used with the *imgPixelHeight* property to retrieve the actual size of an image in pixels.

The image object's height and width properties can then be set to display the image at its actual size without either clipping part of the image or scaling the image to fit the image object.

indent

Topic group Related topics

The horizontal indent, in pixels, for each level of tree items.

Property of

TreeView

Description

The *indent* property reflects the amount of indent, in pixels, for each level of tree items, as indicated by the tree item's *level* property. Note that indentation at the root level is also affected by branch lines at the root, which are controlled by the *hasLines* and *linesAtRoot* properties.

inDesign

Topic group Related topics

Whether the object was instantiated normally or by a visual designer.

Property of

Form, Report, SubForm

Description

The Form and Report designers create a special instance of the form or report when designing them. Some actions that occur when the form or report is executed also take place when it is designed, such as the activation of queries. However, other things are missing; the Header is not executed and parameters that are usually passed are not present.

The *inDesign* property is *true* when the object was created for design instead of execution. Use it to take shortcut actions to allow the design of the object without error.

integralHeight

Topic group Related topics

Whether a partial row at the bottom of the grid is displayed.

Property of

Grid

Description

Set *integralHeight* to *true* to show complete rows only. If *true* and the row at the bottom of the grid is clipped, the entire row is hidden. When *false* (the default), the partial row is shown.

isRecordChanged( )

Topic group Related topics Example

Returns a logical value that indicates whether data in the current record buffer has been modified.

Syntax

<oRef>.*isRecordChanged*(<keystroke expC>)

<oRef>

An object reference to the form.

Property of

Form, SubForm

Description

Use *isRecordChanged*( ) for form-based data handling with tables in work areas. When using the data objects, *isRecordChanged*( ) has no effect; check the rowset's *modified* property instead.

Form-based data buffering lets you manage the editing of existing records and the appending of new records. Editing changes to the current record are not written to the table until there is navigation off the record, or until *saveRecord*( ) is called. Each work area has its own separate edit buffer. *isRecordChanged*( ) returns *true* if the any fields in the currently selected work area have changed; otherwise it returns *false*.

key

Topic group Related topics

Event fired when the user types a keystroke in a control; return value may alter or cancel the keystroke.

Parameters

<char expN>

The ASCII value of the character typed.

<position expN>

The position of the new character in the string.

<shift expL>

Whether the Shift key was down.

<ctrl expL>

Whether the Ctrl key was down.

Property of

ColumnEditor, ComboBox, Editor, Entryfield, ListBox, SpinBox

Description

Use *key* to evaluate and possibly modify each character that the user enters in a control, or to perform some action for each keystroke.

The *key* event handler must return a numeric or a logical value. A numeric value is interpreted as the ASCII code of a character, which automatically replaces the character input by the user. A logical value is interpreted as a decision to accept or reject the character input by the user.

Keystrokes simulated by a control's *keyboard( )* method will fire the *key* event, as will the KEYBOARD command when the control has focus.

Note

You cannot trap all keystroke combinations with *key*. Many Alt+ and Ctrl+ key combinations are reserved for menu and operating system commands, such as Ctrl+X and Ctrl+V for standard Windows cut-and-paste, Alt+F4 to close the application window, etc. These and other common shortcut key combinations will not cause a control's *key* event to fire.

keyboard( )

Topic group Related topics

Stuffs a character string into an edit control, simulating typed user input.

Syntax

<oRef>.*keyboard*(<keystroke expC>)

<oRef>

The control to receive the keystrokes.

<keystroke expC>

A string, which may include key codes.

Property of

Browse, ComboBox, Editor, Entryfield, SpinBox

Description

Use *keyboard*( ) when you want to simulate typing keystrokes into a control. The control does not have to be the one that has focus.

Note

If you want to set a value in a control, it's better to assign the *value* property directly.

Use curly braces enclosed by quotation marks ("{ }") in <keystroke expC> to indicate cursor keys or characters by ASCII code. The following key labels may be used inside the curly braces:

Alt+0 through Alt+9	Ctrl+LeftArrow	Enter	RightArrow
Alt+A through Alt+Z	Ctrl+PgDn	Esc	Shift+F1 through Shift+F9
Backspace	Ctrl+PgUp	F1 through F12	Shift+Tabw
Backtab	Ctrl+RightArrow	Home	Space or Spacebar

Ctrl+A through Ctrl+Z	Ctrl+Tab	Ins	Tab
Ctrl+End	Del	LeftArrow	UpArrow
Ctrl+F1 through Ctrl+F10	DnArrow	PgDn	
Ctrl+Home	End	PgUp	

You may specify a character by its ASCII code by enclosing the value in the curly braces. If the value inside the curly braces is not a recognized key label or ASCII value, the curly braces and whatever is between them are ignored.

Calling *keyboard*( ) immediately fires the control's *key* event, if any. In contrast, the KEYBOARD command stuffs keystrokes in the main typeahead buffer. The control that has focus then picks up the keys from the typeahead buffer as usual.

lastRow( )

Topic group Related topics

Returns a bookmark for the row currently displayed in the last row of the grid.

Syntax

<oRef>.*lastRow*( )

<oRef>

A reference to a grid object.

Property of

Grid

Description

Calling the *lastRow*( ) method returns a bookmark to the row currently displayed on the grid's last, or bottom, row. Note that if the grid's *integralHeight* property is set to *false*, the last row may be partly, or mostly, hidden by the bottom border of the grid.

If the grid is not datalinked to any rowset, the *lastRow*( ) method returns null.

See the example in the topic *firstRow*( ). These methods are quite similar and the example may prove helpful.

left

Topic group Related topics

The position of the left edge of an object relative to its container.

Property of

Form, SubForm and all form contained objects.

Description

The unit of measurement in a form or report is controlled by its *metric* property. The default metric for forms is characters, and for reports it's twips.

level

Topic group Related topics

The tree level of the item.

Property of

TreeItem

Description

The tree item's read-only *level* property contains the nesting level of the item. The top level in the tree is level number one.

lineNo

Topic group Related topics

The current line in an Editor object.

Property of

Editor

Description

Use the *lineNo* property to move the cursor to a specified line in an Editor object, or to determine what line the cursor is on.

When you set the *lineNo* property, the cursor moves to the beginning of the specified line.

linesAtRoot

Topic group Related topics

Whether a line connects the tree items at the first level.

Property of

TreeView

Description

Set *linesAtRoot* to *false* to disable the connecting branch lines at the first level of the tree. To disable all branch lines, set *hasLines* to *false*.

linkFileName

Topic group Related topics

Identifies which OLE document file (if any) is linked with the current OLE field when that field is displayed in an OLE viewer.

Property of

OLE

Description

Use *linkFileName* to identify which OLE document file is linked with the current OLE field when that field is displayed in an OLE viewer. *linkFileName* is a read-only property.

loadChildren( )

Topic group Related topics

Loads and instantiates TreeItems from a text file.

Syntax

<oRef>.*loadChildren*(<filename expC>)

<oRef>

The TreeView object to contain the TreeItems.

<filename expC>

The name of the text file containing the TreeItem objects and properties.

Property of

TreeView

Description

Use *loadChildren*( ) to load TreeItem object definitions and properties from a text file, and instantiate them as the children of an existing TreeView object. The file containing the TreeItems may have been created in a text editor, or by any TreeView object's *streamChildren( )* method.

loadChildren( ) releases all existing child TreeItems in the TreeView and replaces them with the TreeItems in the text file.

lockedColumns

Topic group Related topics

The number of columns that remain locked on the left side of the grid as it is scrolled horizontally.

Property of

Browse, Grid

Description

The *lockedColumns* property specifies the number of contiguous columns on the left side of the grid that do not move when you are scrolling the grid. The number must be between zero and the number of columns in the grid. When the grid is scrolled horizontally, the number of columns you locked will remain displayed in the same position. Note that if you allow column moving, the user can rearrange the columns; whatever columns end up on the left are locked.

Set *lockedColumns* to zero to unlock all columns.

Any change in the *lockedColumns* property will reset the *firstColumn* property to one.

maximize

Topic group Related topics

Determines if a form can be maximized when it's not MDI.

Property of

Form, SubForm

Description

Set *maximize* to *false* to disable the maximize icon and the Maximize option in the system menu. You must set *maximize* before you open the form. If both *maximize* and *minimize* are *false*, their icons do not appear in the title bar. If either one is *true*, they both appear, with one of them disabled.

minimize has no effect unless the form's *mdi* property is *false*; if it is *true*, the form follows the MDI specification and has its maximize icon enabled.

maxLength

Topic group Related topics

Specifies the maximum number of characters allowed in a combobox or entryfield.

Property of

Combobox, Entryfield

Description

When a Combobox or Entryfield control is *dataLink*ed to a field, the control automatically reads the length of the field into its *maxLength* property to set the maximum number of characters allowed.

You may set *maxLength* manually to override this setting, or when using a combobox or entryfield that is not *dataLink*ed to a field. If you set the *maxLength* longer than the field, the entry is allowed, but the field value will be truncated when the value is stored in the table.

mdi

Topic group Related topics

Determines if a form conforms to the Multiple Document Interface (MDI) standard.

Property of

Form

Description

MDI is a Windows specification for opening multiple document windows within the application window. Most word processors are MDI applications. In *dBASE Plus*, all windows are forms. MDI forms have the following characteristics:

- Like application windows, they are moveable and sizeable.
- They are listed on the Windows menu of the application.
- They have a title bar, and in that title bar are the system menu, minimize, maximize, and close icons.
- When one MDI form is maximized, all other MDI forms in the same application are maximized.
- When they are active, their menus replace the menus in the main menu bar.
- They cannot be modal.
- The shortcut keystroke to close the form is Ctrl+F4.

The opposite of MDI is SDI (Single Document Interface), where each document is in its own application window. The Windows Explorer is an SDI application. SDI forms have the following features:

- They each have complete control over their appearance; whether they are movable, sizeable, have a title bar, or any control icons enabled.

Each form is listed separately in the Windows Taskbar.
Their menus appear in the form.
They can be modeless or modal.
They can be set to always display on top of other windows, or appear as palette windows.
The shortcut keystroke to close the form is Alt+F4.

A form's *mdi* property determines whether a form is MDI or SDI. When *mdi* is *true*, the following properties are ignored:

maximize
minimize
moveable
sizeable
smallTitle
sysMenu
topMost

Those properties default to the corresponding values for an MDI form.

Because an MDI form cannot be modal, you cannot open an MDI form with the *readModal*( ) method.

memoEditor

Topic group Related topics

A reference to a control's memo editor object.

Property of

Entryfield

Description

When editing a memo field with a *dataLink*ed entryfield, you may open an Editor object in another form by double-clicking the entryfield, or by calling the entryfield's *showMemoEditor*( ) method.

When the editor is open, the entryfield's *memoEditor* property contains a reference to that Editor object. You may use the *memoEditor* property to manipulate the editor, or close the form that contains it. The memo editor is a standard Editor object, *anchor*ed to a form, so the form is the *memoEditor* object's *form* and *parent* object.

menuFile

Topic group Related topics

Assigns a menubar to a form.

Property of

Form

Description

Use *menuFile* to designate the menu that is displayed when the form has focus. If a form's *menuFile* property is empty, a default menu is displayed when the form has focus.

Menubars created by the Menu designer are stored in .MNU files, which is the default extension for file names assigned to *menuFile*. Assigning a file to *menuFile* executes the named file with

the form as the parameter. The default bootstrap code for a .MNU file creates a menu named root as a child of the form. The file assigned to *menuFile* is automatically loaded as a procedure file. The procedure file's reference count is decremented when the form is released; if that was the last form that used that menu file, it is automatically unloaded.

metric

Topic group Related topics Example

The units of measurement for the position and size of an object.

Property of

Form, Report, SubForm, all form objects.

Description

metric is an enumerated property that can have the following values:

Value	Description
0	Chars (default)
1	Twips
2	Points
3	Inches
4	Centimeters
5	Millimeters
6	Pixels

The Chars *metric* is based on the average height and width for characters in a specific base font. The base font is set through the form's *scaleFontBold*, *scaleFontName*, and *scaleFontSize* properties.

All position and size properties such as *top*, *left*, *height*, and *width* are expressed in the form's current *metric* units. The form's *metric* cannot be changed once the form is opened. If the form is closed, changing the *metric* scales the position and size properties of all the components on the form.

When a control is saved as a custom control, the metric of the form is saved with the control definition. This way, when the control is dropped on another form, assigning the original metric of the control will resize the control appropriately on the new form. In order for this to work properly the custom control's metric property must be set after the control's left, top, height, and width properties are set.

Anytime a control's metric property is set (rather than a form's metric property), dBASE will compare the control's metric with its parent container's metric (usually its form). If they are different, dBASE will convert the control's left, top, height, and width values of the control from the control's metric to the parent container's metric.

minimize

Topic group Related topics

Determines if a form can be minimized when it's not MDI.

Property of

Form, SubForm

Description

Set *minimize* to *false* to disable the minimize icon and the Minimize option in the system menu. The form can still be minimized in other ways, such as choosing Minimize All Windows from the context menu of the Windows Taskbar. You must set *minimize* before you open the form. If both *maximize* and *minimize* are *false*, their icons do not appear in the title bar. If either one is *true*, they both appear, with one of them disabled.

minimize has no effect unless the form's *mdi* property is *false*; if it is *true*, the form follows the MDI specification and has its minimize icon enabled.

modify

Topic group Related topics

Determines if the user can alter data in a Browse or Editor object.

Property of

Browse, Editor

Description

Set *modify* to *false* when you want to make a control read-only.

mousePointer

Topic group Related topics

Changes the appearance of the mouse pointer.

Property of

Most form objects

Description

Use *mousePointer* to provide a visual cue when the user moves the mouse pointer over an object. For example, one pointer style might mean an object is disabled, while another pointer style might mean the object is ready for input.

You can specify the following settings for *mousePointer*:

0 (Default)	N/A	7 (Size S)	
1 (Arrow)		8 (Size NWSE)	
2 (Cross)		9 (Size E)	
3 (I-Beam)		10 (UpArrow)	
4 (Icon)		11 (Wait)	

5 (Size)		12 (No)	
6 (Size NESW)		13 (Hand)	

move( )

Topic group Related topics Example

Repositions and resizes an object.

Syntax

<oRef>.*move*(<left expN> [, <top expN> [, <width expN> [, <height expN>]]])

<oRef>

The object to move or resize.

<left expN>

The new *left* property.

<top expN>

The new *top* property.

<width expN>

The new *width* property. To change the size of the image, you must specify both the <left expN> and the <top expN>.

<height expN>

The new *height* property.

Property of

Most form objects

Description

Use *move*( ) to reposition and/or resize an object in one step. You could assign the four properties directly, but doing so would require four separate steps, and the object would have to be moved and/or resized after each step. Using *move*( ) is faster.

If you want to resize the object without moving it, pass the current *left* and *top* properties as parameters to *move*( ), along with the new width and height.

If you're using *move*( ) to resize an image, the object's *alignment* property should be set to either Stretch (0) or Keep Aspect Stretch (3).

moveable

Topic group Related topics

Determines if a form can be moved when it's not MDI.

Property of

Form, SubForm

Description

Set *moveable* to *false* to prevent the form from being moved in the usual manner. Dragging the title bar has no effect, and the Move option in the system menu is disabled. However, if *sizeable* is *true*, you can move the edges of the form and in-effect move the form.

sizeable has no effect unless the form's *mdi* property is *false*; if it is *true*, the form follows the MDI specification and is always resizeable.

multiple

Topic group Related topics

Specifies whether a ListBox object allows selection of more than one item at a time, or whether a NoteBook object can have more than one row of tabs.

Property of

ListBox, NoteBook

Description

Set *multiple* to *true* if you want to allow the selection of more than one item at one time in a ListBox object. The selections—whether there's one, many, or none—are returned by the ListBox object's *selected*( ) method.

If a NoteBook object's *multiple* property is *false*, all its tabs are displayed in a single row. If there are more tabs than will fit in the width of the notebook, scroll arrows appear. If you set *multiple* to *true*, the tabs are stacked, taking up as many rows as needed, decreasing that amount of space below the tabs. The notebook's *visualStyle* property has more effect when *multiple* is *true*.

multiSelect

Topic group Related topics

Whether multiple rows are visually selected.

Property of

Grid

Description

multiSelect is like *rowSelect*, except that you can select multiple rows. Use the *selected*( ) method to get the bookmarks for the rows that have been selected.

name [Form]

Topic group Related topics

The name of the form property that is used to refer to a component.

Property of

All form components

Description

A component's *name* property reflects the name of the property of the form that is used to refer to the component.

For example, if pushing one button makes another button visible, the code looks like this:

```
function oneButton_onClick( )
form.anotherButton.visible = true
```

In oneButton's event handler, *form* refers to the form that contains the button, and anotherButton is a property of the form that contains an object reference to the PushButton object anotherButton.

When the form was created in the Form designer, the *name* property of the PushButton object was set to anotherButton. When the form is saved into a .WFM file, the resulting code for the button looks like this:

```
this.anotherButton = new PushButton(this)
with (this.anotherButton)
left = 10
top = 0
width = 8
endwith
```

The name of the button is never assigned to the *name* property. Instead, the name of the button is determined by the name of the form property that contains the reference to the object. This is true for any form component that has a *name* property.

To change the name of a component in the .WFM file, change the name of the property in the initial assignment statement and the WITH statement below it.

When you read a component's *name* property, *dBASE Plus* returns the name of the property that the component's *parent* (the form unless the component is in a Container or NoteBook object) uses to refer to the object. The *name* is always all-uppercase.

If you assign a value to a component's *name* property, you actually change the name of the form property that contains the component's object reference. While this is allowed, there aren't many reasons you would want to do that—avoid it.

nativeObject

Topic group Related topics Example

The object that contains the native properties, events, and methods of the ActiveX control.

Property of

ActiveX

Description

An ActiveX object's *nativeObject* property contains a reference to an object that contains the properties, events, and methods, of the actual ActiveX control. Placing the native properties in a separate object prevents name conflicts between the properties of the *dBASE Plus* ActiveX object, and any ActiveX control it represents.

The *nativeObject* object is empty until the *classId* property is set.

nextObj

Topic group Related topics Example

The object that is going to get focus during a focus change.

Property of

Form, SubForm

Description

nextObj contains a reference to the control that is going to get focus during a focus change, for example when you click on another control or press Tab or Shift+Tab. If no focus change is pending, *nextObj* is *null*.

nextSibling

Topic group Related topics Example

The next tree item with the same parent.

Property of

TreeItem

Description

The read-only *nextSibling* property contains an object reference to the next tree item (down) that has the same parent. If the tree item is the last one, *nextSibling* is *null*.

Use *nextSibling* to loop forward through the items in a tree (or subtree).

noOfChildren

Topic group Related topics

The number of child tree items.

Property of

TreeItem

Description

The read-only *noOfChildren* property contains the number of children a tree item has. It goes down one level only; it does not count grandchildren.

oleType

Topic group Related topics

Returns a number that reveals whether an OLE field is empty, contains an embedded document, or contains a link to a document file.

Property of

OLE

Description

Use *oleType* to determine the state of an OLE field. It is a read-only property that may have one of the following three values:

Value	Description
0	Empty
1	Document link

2 Embedded document

onAppend

Topic group Related topics

Event fired when a record is added to a table.

Parameters

none

Property of

Browse, Form, SubForm

Description

The form's (or browse's) *onAppend* event is used mainly for form-based data handling with tables in work areas. It also fires when the *onAppend* event of the form's primary rowset fires.

Use *onAppend* to make your application respond each time the user adds a record. *onAppend* fires after the new record is saved. If the record is saved because the user navigated to another record, *onAppend* fires after arriving at the other record, before *onNavigate*.

onAppend will not work unless the form is open and has controls *dataLink*ed to fields. For example, if you USE a table, create and open an empty Form, assign an *onAppend* event handler, and APPEND BLANK, the *onAppend* will not fire simply because the form is open.

onCellPaint

Topic group Related topics Example

An event fired right after a grid cell is painted.

Parameters

<bSelectedRow>

bSelectedRow is *true* if the grid cell being painted is part of a selected row. Otherwise bSelectedRow is *false*

Property of

ColumnCheckBox, ColumnComboBox, ColumnEditor, ColumnEntryField, ColumnHeadingControl, ColumnSpinBox

Description

Use the *onCellPaint* event to change the settings of a GridColumn's *editorControl* or *headingControl* just after the control is used to paint a grid cell.

The *onCellPaint* event should be used after a *beforeCellPaint* event has changed the properties of a GridColumn's *editorControl* or *headingControl*. You must use the *onCellPaint* event to set the control back to its prior state or to its default state. Otherwise, the changes made in the *beforeCellPaint* event will affect the other cell's within the same grid column.

Using *onCellPaint*

In order to use *onCellPaint*, a grid must be created with explicitly defined GridColumn objects (accessible through the grid's columns property).

In an *onCellPaint* event handler, you can change an editorControl's or headingControl's properties based (optionally) on the current value of the cell. Within *onCellPaint*, the current cell value is contained in this.value.

Initializing a Grid that uses *onCellPaint*

When a form opens, a grid on the form is usually painted before the code setting up any *onCellPaint*( ) event handlers is executed. Therefore, you should call the grid's *refresh*( ) method from the form's *onOpen* event to ensure the grid is painted correctly when the form opens.

Warning: The grid's painting logic is optimized to only load an editorControl's value when it needs to paint it, or give it focus. This means the value loaded into other column's editorControls may not be from the same row as the one used for the currently executing *onCellPaint* event. You should, instead, use the values from the appropriate rowset field objects in order to ensure you are using values from the correct row.

onChange

Topic group Related topics Example

When the contents of the component have been changed.

Parameters

none

Property of

Browse, Checkbox, ComboBox, Editor, Entryfield, Form, ListBox, Ole, RadioButton, ScrollBar, Slider, SpinBox, Subform, Treeview

Description

onChange fires when the user changes data, which includes the following actions:

- Inserts or removes a checkmark in a checkbox
- Selects a different RadioButton in the RadioButton group
- Selects a different item in a tree
- Changes a value in an entryfield
- Changes a value in the text box portion of a spinbox
- Clicks the spinner on a spinbox
- Moves the scroll thumb in a scrollbar object
- Changes a value in a field and moves to another row in a browse
- Changes a value in the test box portion of a combobox.
  - If the value is changed by editing the value, onChange fires when the combobox loses focus.
  - If the value is changed by navigating to a different item in the combobox's list or dropdown list, onChange fires immediately.
- The *onChange* event of an OLE object fires each time the record pointer moves from one record to another. The *onChange* event of a form fires after moving to another record, if the previous record was changed, but only if the form is open and has controls *dataLink*ed to fields.

onChangeCancel

Topic group Related topics

Fires when the user takes an action that closes the dropdown list without choosing an item from the list for a style 1 or 2 combobox

Parameters
none

Property of
Combobox

Description
onChangeCancel fires when the dropdown list closes in the following situations:

- When the user left clicks the mouse anywhere except on the dropdown list window or the combobox dropdown button
- When the user presses the tab key or escape key while the dropdown list is open
- When the user left clicks on the Close button for the form containing the combobox

onChangeCancel can be used to detect that the user has closed the dropdown list without clearly and unambiguously choosing an item from the list. In some situations it may be necessary to detect this and, possibly, set the combobox value back to a previous value since the current combobox value may have been changed due to the user navigating through the dropdown list.

When the dropdown list closes, either onChangeCancel will fire or onChangeCommitted will fire, not both.

onChangeCommitted

Topic group Related topics
Fires when the user takes an action to unambiguously choose an item from the list.

Parameters
none

Property of
Combobox

Description
onChangeCommitted will fire in the following cases:

- Left click on an item in the listbox (all styles) when the item is different from the current ComboBox value.
- Press Enter with an item highlighted in the dropdown list (style 1 or 2) when the item is different from the current ComboBox value.
- For style 0 or for style 1 or 2, with the dropdown list closed, press the Up Arrow, Down Arrow, PgUp, or PgDn keys.
- Left click on the ComboBox button for a style 1 or 2 ComboBox when the dropdown list is open and the highlighted item in the dropdown list is different from the current ComboBox value.

onChangeCommitted() will not fire for a style 1 or 2 ComboBox when the dropdown list is open and the Up Arrow, Down Arrow, PgUp, or PgDn keys are pressed. (Note that this is different from the onChange() event which does fire in these cases).

onChangeCommitted() fires only after the ComboBox's value property has been updated with the selected value.

When the dropdown list closes, either onChangeCommitted will fire or onChangeCancel will fire, not both.

onChar

Topic group Related topics

Event fired when a "printable" key or key combination is pressed while the control has focus.

Parameters

<char expN>

The ASCII code of the key or key combination

<repeat count expN>

The number of times the keystroke is repeated based on how long the key is held down.

<key data expN>

A double-byte value that contains information about the key released, stored in separate bit fields. The bits of this parameter contain the following information:

Bit numbers	Description
0–7	Keyboard scan code (OEM dependent)
8	Extended key (1 if *true*) such as right Alt and Ctrl, and numbers on numeric keypad
9–12	Reserved
13	Context code: 1 if Alt was pressed during keystroke
14	Previous key state: 1 if key was held down
15	Transition state: 1 if key is being released, 0 if key is pressed (usually 0)

Property of

PaintBox

Description

If you have created a PaintBox object to develop a custom edit control, use *onChar* to do something when the object has focus and the user presses a key; that is, when they type a normal character.

onChar is similar to *onKeyDown*. However, *onChar* doesn't fire for non-printable keys, such Caps Lock, while *onKeyDown* fires for any key pressed.

onCheckBoxClick

Topic group Related topics

Event fired after a checkbox in a tree item is clicked.

Parameters

none

Property of

TreeView

Description

onCheckBoxClick fires after the user has clicked a tree item's checkbox. Check the *checked* property of the tree's currently *selected* tree item to see whether the checkbox is now checked or not.

onClick

Topic group Related topics Example

After a button is clicked.

Parameters

none

Property of

Menu, PushButton

Description

Use *onClick* to execute code when you click a button or choose a menu item.

onClose

Topic group Related topics

After the form has been closed.

Parameters

none

Property of

Form, OLE, PaintBox, SubForm

Description

Use *onClose* to perform any extra manual cleanup, if necessary, when you close a form or report. Normally, *dBASE Plus* automatically discards anything in the form when you close it. You might use *onClose* if you created an object in the *onOpen* that you did not bind to the form or report.

Before executing the *onClose* event handler, *dBASE Plus* does the following:

1. Executes the *canClose* event handler (if any) of the form. If it returns *false*, the form does not close; nothing further happens.
2. Executes the *valid* event handler (if any) of the object that currently has input focus. If it returns a value of *false*, the form does not close; nothing further happens.
3. Executes the *onLostFocus* event handler (if any) of the object that currently has input focus.
4. Executes the *onLostFocus* event handler (if any) of the form.

The *onClose* events of an OLE control executes when the parent form is closed, after the *onClose* of that form.

onDesignOpen

Topic group

After a form or component is loaded in the Form designer.

Parameters

<from palette expL>

Whether the component was added from the palette. If *true*, the component has just been created. If *false*, the component has been reloaded into the Form designer (when editing an existing form).

Property of

All form objects.

Description

Use *onDesignOpen* to execute code whenever a form or component is loaded into the Form Designer, either when it is first created (for components only), or when it is subsequently loaded into the Form Designer.

onDragBegin

Topic group Related topics

For Drag&Drop operations; when a drag operation actually begins.

Parameters

(none)

Property of

Many Form objects

Description

Use *onDragBegin* to perform actions when a drag operation commences for a Drop Source object.

The *onDragBegin* event only fires when a Drag&Drop operation is initiated by a Drop Source object's *drag*( ) method.

onDragEnter

Topic group Related topics Example

For Drag&Drop operations; when the mouse enters the display area of an active Drop Target.

Parameters

<nLeft expN>

The entry position of the mouse pointer relative to the left edge of the Drop Target object.

<nTop expN>

The entry position of the mouse pointer relative to the top edge of the Drop Target object.

<cType expC>

A character or string, typically identifying the dragged object's type.

<cName expC>

A string, typically containing the name of an object or a file.

Property of

Browse, Container, Form, Grid, Image, ListBox, NoteBook, PaintBox, ReportViewer, SubForm, TreeView

Description

Use *onDragEnter* to perform actions when the mouse enters the display area of an active Drop Target during a Drag&Drop operation.

A numeric value returned by the *onDragEnter* event handler determines whether a drop will be allowed, or may change the type of drop operation. The permitted return values are:

Value	Drop Effect
0	No drop permitted
1 (default)	Drop permitted: Copy
2	Drop permitted: Move

If *onDragEnter* is not explicitly defined for a Drop Target object or no value is returned, a default value of 1 is assumed.

Note

onDragEnter is not invoked for files dragged from the *dBASE Plus* Navigator window.

onDragLeave

Topic group Related topics

For a Drag&Drop operation; when the mouse leaves an active Drop Target object's display area without having dropped anything.

Parameters

(none)

Property of

Browse, Container, Form, Grid, Image, ListBox, NoteBook, PaintBox, ReportViewer, SubForm, TreeView

Description

Use *onDragLeave* to perform actions when the mouse leaves the display area of an active Drop Target object during a Drag&Drop operation.

The *onDragLeave* event only fires when a drop was allowed, but not actually performed.

Note

onDragLeave is not invoked for files dragged from the *dBASE Plus* Navigator window.

onDragOver

Topic group Related topics

For Drag&Drop operations; event fired while the mouse drags an object over an active Drop Target object's display area.

Parameters

<nLeft expN>

The position of the mouse relative to the left edge of the Drop Target object.

<nTop expN>

The position of the mouse relative to the top edge of the Drop Target object.

<cType expC>

A character or string, typically identifying the dragged object's type.

<cName expC>

A string, typically containing the name of an object or a file.

Property of

Browse, Container, Form, Grid, Image, ListBox, NoteBook, PaintBox, ReportViewer, SubForm, TreeView

Description

Use *onDragOver* to perform actions while an object is being dragged over the display area of an active Drop Target object. This allows you to control whether or not the dragged object may be dropped at specific mouse cursor locations.

A numeric value returned by the *onDragOver* event handler determines whether a drop is allowed, or may change the type of drop operation allowed at the specified cursor position. The permitted return values are:

Value	Drop Effect
0	No drop permitted
1 (default)	Drop permitted: Copy
2	Drop permitted: Move

onDragOver will not be fired if the Drop Target's *onDragEnter* event handler was invoked and returned zero (no drop permitted).

If an *onDragOver* event handler is not defined for a Drop Target object or no value is returned, a default value of 1 is assumed.

Note

onDragOver is not invoked for files dragged from the *dBASE Plus* Navigator window.

onDrop

Topic group Related topics Example

For Drag&Drop operations; when the mouse button is released over an active Drop Target object during a Drag&Drop Copy operation.

Parameters

<nLeft expN>

The position of the dropped object relative to the left edge of the Drop Target object.
The editor's current columNo character position.

<nTop expN>

The position of the dropped object relative to the top edge of the Drop Target object.
The editor's current lineNo position.

<cType expC>

A character or string, typically identifying the dropped object's type.
If a file is being dropped onto the editor, this parameter will contain an "F".
If text is being dropped onto the editor, this parameter will contain a "T".

<cName expC>

A string, typically containing the name of an object or a file.
The filename or text being dropped onto the editor.

Property of

Browse, Container, Editor, Form, Grid, Image, ListBox, NoteBook, PaintBox, ReportViewer, SubForm, TreeView

Description

Use *onDrop* to perform actions when the mouse button is released over an active Drop Target object during a Drag&Drop Copy operation. The *onDrop* event does not fire for a Move operation.

When a Copy operation is initiated from a Drop Source object's *drag*( ) method, <cType expC> and <cName expC> will contain the parameter strings passed by the method.

Files may be dragged from the *dBASE Plus* Navigator, or from any Windows® OLE Drag and Drop compliant application. What is received in <cType expC> depends on which drag-initiating application is used (i.e. Explorer, WinZIP, etc), but will usually be "F" for a file. <cName expC> will usually contain the filename.

When multiple files are selected and dragged, *onDrop* will fire multiple times in succession, once for each file in the selected block.

Note:

When files are dragged from the *dBASE Plus* Navigator window, the *onDragEnter*, *onDragOver*, and *onDragLeave* events will not fire. However, these events will fire when files are dragged from OLE Windows applications.

Note:

Mouse button "Up" events are "consumed" by *onDrop* events. This prevents, for example, the firing of a Drop Target object's *onLeftMouseUp* event when the left mouse button is used for a Drag&Drop Copy operation. If the drop fails, the *onDrop* must explicitly call the mouse button Up handler function.

The Editor class and onDrop

When FALSE is returned by *onDrop*, the drop will not occur.

When TRUE is returned by *onDrop*, and a file is being dropped, the file's contents will be inserted into the editor starting at:

```
lineNo = nTop+1, columnNo = 1
```

When TRUE is returned by *onDrop*, and text is being dropped, the text will be inserted into the editor starting at:

```
lineNo = nTop, columnNo = nLeft
```

onEditLabel

Topic group Related topics Example

Event fired after the text label in a tree item is edited; may optionally return a different label value to save.

Parameters

<text expC>

The not-yet-posted text label.

Property of

TreeView

Description

onEditLabel fires after the user has pressed Enter or clicked away to submit their label change.

If the *onEditLabel* event handler returns a character string, that string is saved as the *text* property of the tree item instead of <text expC>. If the event handler returns any other type, or returns nothing, <text expC> is used as-is.

onEditPaint

Topic group Related topics

For a style 0 or 1 combobox, fires for each keystroke that modifies the value of the combobox, just after the new value is displayed.

Parameters

none

Property of

Combobox

Description

For a style 0 or 1 combobox, fires for each keystroke that modifies the value of the combobox, just after the new value is displayed.
onEditPaint() fires just after displaying the new value for a ComboBox. It does not fire if the keystroke does not modify the ComboBox.

onExpand

Topic group Related topics

Event fired after a checkbox in a tree item is clicked.

Parameters

<oItem>

The TreeItem whose + or - has been clicked.

Property of

TreeView

Description

The *onExpand* event fires after the user has expanded or collapsed a tree item's subtree through the user interface, usually by clicking the + or - icon. Check the *expanded* property of the tree's currently *selected* tree item to see whether the subtree is now expanded or not.

onFormSize

Topic group Related topics

Event fired whenever the parent form of a Grid or PaintBox object is resized.

Parameters

none

Property of

Grid, PaintBox

Description

The *onFormSize* event fires whenever the parent form of a Grid or PaintBox object is resized, restored, or maximized. This lets you reposition or resize the object based on the form's new size. For example, you could use *onFormSize* to implement behavior similar to the *anchor* property, keeping the bottom of a PaintBox object positioned at the bottom of the form.

For PaintBox objects, the *onFormSize* event is similar to *onPaint*. However, the *onPaint* event is triggered when the parent form is opened, or items covering the PaintBox object are moved away.

onGotFocus

Topic group Related topics

Event fired when a component gains focus.

Parameters

none

Property of

Form and all form components that get focus

Description

onGotFocus fires whenever the form or component gains focus.

The firing order for Events when opening a form:

1. The form's *open( )* method is called
   - The *when* event for each control is fired according to the z-order
   - The form's *onOpen* event is fired
   - The *onOpen* event for each control is fired according to the z-order

The firing order for Form object Events:

Clicking on a form object will result in the following events firing in this order;

2. The *when* event
   - The *onGotFocus* event
   - A mouse event such as, *onLeftDblClick*

Navigating to a form object by using the Tab key will result in the following events firing in this order;

3. The *when* event
   - The *onGotFocus* event

Tip: ON KEY LABEL TAB <command> will perform an action (<command>) when the user presses the TAB key (See ON KEY). However, even though ON KEY LABEL TAB is set to perform <command>, pressing Shift-Tab will still move to another form object (the preceding one in the z-order) and fire it's events in the above order.

onHelp

Topic group Related topics Example

Event fired when the user presses F1 while an object has focus, instead of context-sensitive help.

Parameters

none

Property of

Most form objects

Description

Use *onHelp* to override the built-in context-sensitive help system (based on the *helpFile* and *helpId* properties) and execute your own code when the user presses F1. For example, you might use *onHelp* if you have not yet written a Help file, if the help you want to give is very simple, or you want *dBASE Plus* to drive the help (as you would with an online assistant).

As with context-sensitive help, if you assign an *onHelp* event handler to a form, that is the default handler for all the controls in the form. Each control may then have its own *onHelp* if necessary; otherwise, the form's *onHelp* is fired when the user presses F1.

onKey

Topic group Related topics Example

Event fired after a keypress has been processed for a control.

Parameters

<char expN>

The ASCII value of the key pressed, or the value returned by the *key* event.

<position expN>

The current position of the cursor in the control.

<shift expL>

Whether the Shift key was down.

<ctrl expL>

Whether the Ctrl key was down.

Property of

ComboBox, Entryfield, ListBox, SpinBox

Description

Use *onKey* to evaluate the contents of a control, or to perform some action after each keystroke has been processed either by the control's *Key* event, or the operating system.

Keystrokes simulated by a control's *keyboard*( ) method will fire the *onKey* event, as will the KEYBOARD command when the control has focus.

Note

You cannot trap all keystroke combinations with *onKey*. Many Alt+ and Ctrl+ key combinations are reserved for menu and operating system commands, such as Ctrl+X and Ctrl+V for standard Windows cut-and-paste, Alt+F4 to close the application window, etc. These and other common shortcut key combinations will not cause a control's *onKey* event to fire.

onKeyDown

Topic group Related topics

Event fired when any key is pressed while the control has focus.

Parameters

<virtual key expN>

The Windows virtual-key code of the key released. For a list of virtual-key codes, see the Win32 Programmer's Reference (search for "Virtual-key Codes" in the index).

<repeat count expN>

The number of times the keystroke is repeated based on how long the key is held down.

<key data expN>

A double-byte value that contains information about the key released, stored in separate bits. The bits of this parameter contain the following information:

Bit numbers	Description
0–7	Keyboard scan code (OEM dependent)
8	Extended key (1 if *true*) such as right Alt and Ctrl, and numbers on numeric keypad
9–12	Reserved
13	Context code: always 0 for *onKeyDown*
14	Previous key state: 1 if key was held down
15	Transition state: always 0 for *onKeyDown*

Property of

PaintBox

Description

Use *onKeyDown* and *onKeyUp* for complete control of keystrokes while a PaintBox object has focus. Each key is treated separately, with none of their normal relationships, and pressing and releasing the key are two separate actions. For example, holding down Shift and pressing the A key is normally interpreted as a capital "A". With *onKeyDown* and *onKeyUp*:

onKeyDown fires when the Shift is pressed
onKeyDown continues to fire as the Shift is held down
onKeyDown fires when the A is pressed
onKeyDown continues to fire if the A is held down
onKeyUp fires when the A is released
Releasing a key stops the repeat action of *onKeyDown* for the Shift key
onKeyUp fires when the Shift is released

To know that this was a capital "A", you would have to keep track of the fact that the Shift key was down when the A key was pressed.

A similar event, *onChar* is used when you want the PaintBox to respond to normal "printable" characters. For example, *onChar* would fire just once, getting the ASCII code for the capital "A". *onKeyDown* and *onKeyUp* deal with Windows virtual-key codes, which are not the same as the key character value in many cases.

onKeyUp

Topic group Related topics

Event fired when any key is released while the control has focus.

Parameters

<virtual key expN>

The Windows virtual-key code of the key released. For a list of virtual-key codes, see the Win32 Programmer's Reference (search for "Virtual-key Codes" in the index).

<repeat count expN>

The number of times the keystroke is repeated based on how long the key is held down; always 1 for *onKeyUp*.

<key data expN>

A double-byte value that contains information about the key released, stored in separate bits. The bits of this parameter contain the following information:

Bit numbers	Description
0–7	Keyboard scan code (OEM dependent)
8	Extended key (1 if *true*) such as right Alt and Ctrl, and numbers on numeric keypad
9–12	Reserved
13	Context code: always 0 for *onKeyUp*
14	Previous key state: always 1 for *onKeyUp*
15	Transition state: always 1 for *onKeyUp*

Property of

PaintBox

Description

Use *onKeyUp* with *onKeyDown* for complete control of keystrokes while a PaintBox object has focus. For more information, see *onKeyDown*.

onLastPage

Topic group

Event that fires right after the last page of a report has been rendered.

Parameters

none

Property Of

reportViewer

Description

The *onLastPage* event can be used to:

Detect that a report's last page has been reached.

Provide a place to insert code that would enable, or disable, the appropriate toolbar and menu options.

onLeftDblClick

Topic group Related topics Example

Event fired when the user double-clicks a form or an object.

Parameters

<flags expN>

A single-byte value that tells you which other keys and mouse buttons were pressed when the user double-clicked the button.

<col expN>

The horizontal position of the mouse when the user double-clicked the button.

<row expN>

The vertical position of the mouse when the user double-clicked the button.

Property of

Most form objects

Description

Use *onLeftDblClick* to perform an action when the user double-clicks with the left mouse button. *onLeftDblClick* can also trap Shift, Ctrl, middle mouse button, or right mouse button presses if they occur at the same time the user double-clicks the button.

You can test the state of multiple keys that have been pressed simultaneously. The state of each of the three mouse buttons and the Shift and Ctrl keys is stored in a separate bit in the <flags expN> parameter, as follows:

Bit number	Flag for
0	Left mouse button
1	Right mouse button
2	Shift
3	Ctrl
4	Middle mouse button

To check if the key or button was down, use the BITSET( ) function with the <flags expN> as the first parameter, and corresponding the bit number as the second parameter. BITSET( ) will return *true* if the key or button was down, and *false* if it was not.

The <col expN> and <row expN> parameters contain values that are relative to the object that fired the event. For example, the upper left corner of a button is always row 0, column 0, even if that button is in the bottom corner of the form.

All other onLeft-, onRight-, and onMiddle- mouse events operate in the same way, and receive the same parameters.

When you double-click a button, its button events fire in the following order:

1. mouse down

- mouse up
- mouse double click
- mouse up

onLeftMouseDown

Topic group Related topics Example

Event fired when the user presses the left mouse button while the pointer is over a form or an object.

Description

Use *onLeftMouseDown* to perform an action when the user presses the left mouse button. Other than the initiating action, this event is identical to *onLeftDblClick*.

onLeftMouseUp

Topic group Related topics Example

Event fired when the user releases the left mouse button while the pointer is over a form or an object.

Description

Use *onLeftMouseUp* to perform an action when the user releases the left mouse button. Other than the initiating action, this event is identical to *onLeftDblClick*.

onLostFocus

Topic group Related topics

Event that fires when a component loses focus.

Parameters

none

Property of

Form and all form components that get focus

Description

onLostFocus fires whenever the component loses focus.

onLostFocus differs from *valid*, which specifies a condition that must evaluate to *true* before the object can lose focus.

onMiddleDblClick

Topic group Related topics Example

Event fired when the user double-clicks with the middle mouse button while the pointer is on a form or an object.

Description

Use *onMiddleDblClick* to perform an action when the user double-clicks with the middle mouse button. Other than the initiating action, this event is identical to *onLeftDblClick*.

onMiddleMouseDown

Topic group Related topics Example

Event fired when the user presses the middle mouse button while the pointer is over a form or an object.

Description

Use *onMiddleMouseDown* to perform an action when the user presses the middle mouse button. Other than the initiating action, this event is identical to *onLeftDblClick*.

onMiddleMouseUp

Topic group Related topics Example

Event fired when the user releases the middle mouse button while the pointer is over a form or an object.

Description

Use *onMiddleMouseUp* to perform an action when the user releases the middle mouse button. Other than the initiating action, this event is identical to *onLeftDblClick*.

onMouseMove

Topic group Related topics Example

Event fired when the user moves the mouse over a form or control.

Parameters

<flags expN>

A single-byte value that tells you which keys and mouse buttons were pressed while the user moved the mouse. You can interpret this value with the BITSET( ) function, which examines individual bits in numeric values. For more information, see *onLeftDblClick*.

<col expN>

The horizontal position of the mouse inside the bounds of the object.

<row expN>

The vertical position of the mouse inside the bounds of the object.

Property of

Most form objects

Description

Use *onMouseMove* to perform actions when the user moves the mouse over an object. A control's *onMouseMove* fires when the mouse moves over that control. The form's *onMouseMove* fires when the mouse moves over an area of the form where there is no control.

The <col expN> and <row expN> parameters contain values that are relative to the object that fired the event. For example, the upper left corner of a button is always row 0, column 0, even if that button is in the bottom corner of the form.

onMouseOut

Topic group

Event fired when the user moves a mouse from over a control, form or subform.

Parameters

<flags expN>

A single-byte value that tells you which other keys and mouse buttons were pressed when the mouse was moved from over a control, form or subform.

<col expN>

The horizontal position of the mouse when it was moved from over a control, form or subform.

<row expN>

The vertical position of the mouse when it was moved from over a control, form or subform.

Property of

All form objects, Form, Subform

Description

Use *onMouseOut* to perform an action when the user moves a mouse from over a control, form or subform The *onMouseOut* event can also trap Shift, Ctrl, middle mouse button, or right mouse button presses if they are present at the time the user moved the mouse from over the form, subform or control.

You can test the state of multiple keys that have been pressed simultaneously. The state of each of the three mouse buttons and the Shift and Ctrl keys is stored in a separate bit in the <flags expN> parameter, as follows:

Bit number	Flag for
0	Left mouse button
1	Right mouse button
2	Shift
3	Ctrl
4	Middle mouse button

To check if the key or button was down, use the BITSET( ) function with the <flags expN> as the first parameter, and corresponding the bit number as the second parameter. BITSET( ) will return *true* if the key or button was down, and *false* if it was not.

The <col expN> and <row expN> parameters contain values that are relative to the object that fired the event. For example, the upper left corner of a button is always row 0, column 0, even if that button is in the bottom corner of the form.

onMouseOver

Topic group

Event fired when the user moves a mouse over a control, form or subform.

Parameters

<flags expN>

A single-byte value that tells you which other keys and mouse buttons were pressed when the mouse was moved over a control, form or subform.

<col expN>

The horizontal position of the mouse when it was moved over a control, form or subform.

<row expN>

The vertical position of the mouse when it was moved over a control, form or subform.

Property of

All form objects, Form, Subform

Description

Use *onMouseOver* to perform an action when the user moves a mouse over a control, form or subform The *onMouseOver* event can also trap Shift, Ctrl, middle mouse button, or right mouse button presses if they are present at the time the user moved the mouse over the form, subform or control.

You can test the state of multiple keys that have been pressed simultaneously. The state of each of the three mouse buttons and the Shift and Ctrl keys is stored in a separate bit in the <flags expN> parameter, as follows:

Bit number	Flag for
0	Left mouse button
1	Right mouse button
2	Shift
3	Ctrl
4	Middle mouse button

To check if the key or button was down, use the BITSET( ) function with the <flags expN> as the first parameter, and corresponding the bit number as the second parameter. BITSET( ) will return *true* if the key or button was down, and *false* if it was not.

The <col expN> and <row expN> parameters contain values that are relative to the object that fired the event. For example, the upper left corner of a button is always row 0, column 0, even if that button is in the bottom corner of the form.

onMove

Topic group Related topics

Event fired after the user moves the form.

Parameters

<left expN>

The new horizontal position of the upper left corner of the form’s client area.

<top expN>

The new vertical position of the upper left corner of the form’s client area.

Property of

Form, SubForm

Description

Use *onMove* to perform actions automatically when a form is moved.

The two parameters passed to the event handler indicate the new position of the client area of the form, the area below the title bar and inside the edges of the form. To get the new position of the entire form, check the form's *left* and *top* properties directly.

onNavigate

Topic group Related topics Example

Event fired when the record pointer in a table in a work area is moved.

Parameters

<workarea expN>

The work area number where the navigation took place.

Property of

Browse, Form, SubForm

Description

The form's (or browse's) *onNavigate* event is used mainly for form-based data handling with tables in work areas. It also fires when there is navigation in the form's primary rowset.

Use *onNavigate* to make your application respond each time the user moves from one record to another.

When using tables in work areas, *onNavigate* will not fire unless the form is open and has controls *dataLink*ed to fields. For example, if you USE a table, create and open an empty Form, assign an *onNavigate* event handler, and SKIP in the table, the *onNavigate* will not fire simply because the form is open.

When navigating in the form's primary rowset, the form's *onNavigate* fires after the rowset's *onNavigate*, and the <workarea expN> parameter is zero.

onOpen

Topic group Related topics Example

After the form or component has been opened.

Parameters

none

Property of

All form objects.

Description

onOpen events fire after a form has been opened by either *open*( ) or *readmodal*( ). First the *onOpen* event for the form or report fires, then the *onOpen* for each component, if one has been assigned. Use *onOpen* to set up items in the form that cannot be set in the Form designer.

onPaint

Topic group Related topics

Event fired whenever a PaintBox object needs to be redrawn.

Parameters

none

Property of

PaintBox

Description

onPaint is called whenever a PaintBox object needs to be redrawn. Events that trigger *onPaint* include:

- the parent form is opened
- a minimized parent form is restored or maximized
- a window or object which has been covering the paintbox object is moved away

onRightDblClick

Topic group Related topics Example

Event fired when the user double-clicks with the right mouse button while the pointer is on a form or an object.

Description

Use *onRightDblClick* to perform an action when the user double-clicks with the right mouse button. Other than the initiating action, this event is identical to *onLeftDblClick*.

This event will not fire for the form if you have a popup menu assigned to the form's *popupMenu* property.

onRightMouseDown

Topic group Related topics Example

Event fired when the user presses the right mouse button while the pointer is on a form or an object.

Description

Use *onRightMouseDown* to perform an action when the user presses the right mouse button. Other than the initiating action, this event is identical to *onLeftDblClick*.

If the form has a popup menu assigned to its *popupMenu* property, the sequence of events when you right-click the form is:

1. The popup menu appears
    - After making a choice or dismissing the menu, the *onRightMouseDown* event fires.
    - If a choice was made from the popup menu, its *onClick* fires.

onRightMouseUp

Topic group Related topics Example

Event fired when the user releases the right mouse button while the pointer is on a form or an object.

Description

Use *onRightMouseUp* to perform an action when the user releases the right mouse button. Other than the initiating action, this event is identical to *onLeftDblClick*.

This event will not fire for the form if you have a popup menu assigned to the form's *popupMenu* property.

onSelChange

Topic group Related topics Example

Event fired when a selection is changed in a component.

Parameters

none

Property of

Designer, Grid, ListBox, NoteBook, TabBox

Description

onSelChange is used in components that have a specific set of options to choose from; the tabs in a NoteBook or TabBox, the items in a ListBox, selected objects in a Designer, and the rows or columns in a Grid. It fires whenever the focus changes from one option to another or, in the case of a Grid, when navigation occurs in either rows or columns.

onSelection

Topic group Related topics

Event fired when the user submits a form.

Parameters

<id expN>

The *id* property of the control that had focus when the form was submitted.

Property of

Form, SubForm

Description

A form is submitted when the user either:

Presses Enter when the form has focus and no Editor, Grid, or Browse object has focus.
Presses Spacebar when a pushbutton has focus.
Clicks a pushbutton.

The concept of submitting a form is antiquated and rarely used. You should code the *onClick* event handler for a specific pushbutton, and set the *default* property of a pushbutton to *true* so that pushbutton is clicked when Enter is pressed.

Note

The *default* property will work as described above only when SET CUAENTER is ON. When CUAENTER is OFF, the Enter key emulates the Tab key and merely shifts focus to the next control.

onSize

Topic group Related topics Example

Event fired after the user resizes a form.

Parameters

<nType>

A number that indicates how the user resized the form. It has three possible values:

Value	Description
0	The user resized the form with the mouse or restored the form from a maximized or minimized condition.
1	The user minimized the form.
2	The user maximized the form.

<width>

The new width of the client area of the form.

<height>

The new height of the client area of the form.

Property of

Form, SubForm

Description

Use *onSize* to perform actions when the user resizes a form.

The two parameters passed to the event handler indicate the new size of the client area of the form, the area below the title bar and inside the edges of the form. To get the new size of the entire form, check the form's *width* and *height* properties directly.

Some controls have an *onFormSize* event that fires when the form is resized; any actions specific to those controls should be handled with that event. Other controls have an *anchor* property, and are resized to automatically.

open( )

Topic group Related topics

Opens a form.

Syntax

<oRef>.*open*( )

<oRef>

The form to open.

Property of

Form, SubForm

Description

Use *open*( ) to open a form.

The form you open with *open*( ) is modeless, and has the following characteristics:

While the form is open, focus can be transferred to other forms.
Execution of the routine that opened the form continues after the form is opened and active.

pageCount( )

Topic group Related topics Example

Returns the highest numbered page used in a form.

Syntax

<oRef>.*pageCount*( )

<oRef>

An object reference to the form you want to check.

Property of

Form, SubForm

Description

pageCount( ) returns the highest *pageno* used by the controls in the form. (There are actually no pages or page objects in a form.)

In most cases, you know how many pages there are in a form because you decide on which pages to place the form's controls. Use *pageCount*( ) if you do not want to keep track of the highest page manually, or if the form creates objects on different pages dynamically.

pageno

Topic group Related topics Example

The page of the form on which a component appears, or the form's active page.

Property of

All form objects.

Description

All form objects have a *pageno* property that can be between 0 and 255. The form's *pageno* property indicates the form's active page, the one it is displaying. All the components in the form that have the same *pageno* as the form are displayed on that "page"; the rest are hidden. There are no actual pages or page objects to manage.

When a form's *pageno* property is zero, all components are displayed. If a component's *pageno* property is zero, it appears on all pages. For example, a company logo that appears on every page can be placed on page zero.

The *pageno* property can be changed at any time. Changing a form's *pageno* displays another page of the form. Changing a component's *pageno* moves that component to that page.

In addition to the *pageno* property, you can set a component's *visible* property if you want to hide or display it under particular circumstances.

params

Topic group Related topics Example

Parameters passed to a report.

Property of

ReportViewer

Description

The *params* property contains an associative array that contains parameter names and values, if any, that are passed to the specified .REP file. The parameters are passed in the order they are assigned to the *params* property.

Normally, report parameters are assigned to the *params* array before setting the *filename* property; if they are assigned after setting the *filename* property, you must call the ReportViewer object's *reExecute*( ) method to regenerate the report.

paste( )

Topic group Related topics

Copies text from the Windows clipboard to the control.

Syntax

<oRef>.*paste*( )

<oRef>

An object reference to the control in which to paste the text.

Property of

Browse, ComboBox, Editor, Entryfield, SpinBox

Description

Use *paste*( ) when the user wants to copy text from the Windows clipboard into the specified control.

If you have assigned a menubar to the form, you can use a menu item assigned to the menubar's *editPasteMenu* property instead of using the *paste*( ) method of individual objects on the form.

patternStyle

Topic group Related topics

Specifies the background hatching pattern.

Property of

Rectangle

Description

Use *patternStyle* to select a background hatching pattern for a Rectangle object.

You can specify the following settings for *patternStyle*

Value	Description	Example
0	Solid	
1	BDiagonal	
2	Cross	
3	Diagcross	
4	FDiagonal	
5	Horizontal	
6	Vertical	

The color of the pattern is determined by the foreground and background colors specified in the rectangle's *colorNormal* property.

pen

Topic group Related topics

Specifies the pattern of a Line object.

Property of

Line

Description

Use *pen* to control the appearance of a Line object when its *width* is 1.

You can specify any of five settings for *pen*:

Value	Description	Example
0	Solid	
1	Dash	
2	Dot	
3	Dash Dot	
4	DashDotDot	

If the line's *width* is greater than 1, the *pen* property is ignored and the line is always drawn with a solid pen.

penStyle

Topic group Related topics

Specifies the type of line to be used as the border of a Shape object.

Property of

Shape

Description

Use *penStyle* to control the appearance of the border of a Shape object when the *penWidth* is 1.

You can specify any of five settings for *penStyle*:

Value	Description	Example
0	Solid	
1	Dash	
2	Dot	
3	Dash Dot	
4	DashDotDot	

If *penWidth* is greater than 1, the *penStyle* property is ignored and the outline is always drawn with a solid pen.

penWidth

Topic group Related topics

Specifies the width in pixels of the line used as the border of a Shape object.

Property of

Shape

Description

Use *penWidth* to specify the thickness of the line used to border a shape object. If you set *penWidth* to a value greater than 1, then *penStyle* is always treated as 0 (Solid).

persistent

Topic group

Determines whether custom control, datamodule, menu or procedure files associated with a form are loaded in the persistent mode.

Property of

Form, SubForm

Description

When set to true, the persistent property prevents files from being closed directly, through CLOSE ALL and CLOSE PROCEDURE, or implicitly, through SET PROCEDURE TO. To close a file which has been tagged "persistent", you must include the PERSISTENT designation at the end of the command, i.e, CLOSE ALL PERSISTENT, CLOSE PROCEDURE PERSISTENT, or compile the file using COMPILE <filename>.

When set to true, the *persistent* property has the following effects:

- When the form is being modified in the Form Designer, all SET PROCEDURE TO statements streamed in the form class definition will have the PERSISTENT option specified.
- When the form runs, the form file itself, and any menu and datamodule files it uses, are internally loaded as PERSISTENT.

The execution of explicit SET PROCEDURE TO statements are not affected by the *persistent* property of the form. The presence (or absence) of the PERSISTENT option in any given SET PROCEDURE TO command determines whether it is loaded as "persistent". However, if the Form Designer was used, and Form.Persistent is set to True, all SET PROCEDURE TO statements will be loaded as "persistent".

phoneticLink

Topic group

Contains a reference to the control that mirrors the phonetic equivalent of the current value.

Property of

Entryfield

Description

phoneticLink is used in double-byte operating systems to store the single-byte phonetic representation of a value in an entryfield. It contains an object reference or the *name* of the mirror Entryfield object.

picture

Topic group Related topics Example

A formatting template for text.

Property of

Entryfield, SpinBox, Text

Description

Specify the *picture* property with a character string called a template. A template can consist of

Picture template characters, which represent and modify individual characters in the text string.
Function symbols, which usually modify the entire text string. (For information on function symbols, see the *function* property.)
Literal placeholders (characters other than template codes), which are inserted into the text string.

Here are the *picture* template characters:

9 Restricts entry of character data to numbers. Restricts entry of numeric data to numbers and + and - signs

Restricts entry to numbers, spaces, signs and the SET POINT character.

! Converts letters to uppercase

$ Inserts a dollar sign or the symbol defined with SET CURRENCY TO instead of leading blanks

% Inserts a percent sign as the right-most character of a numeric template

***** Inserts asterisks in place of leading spaces

. Marks the position of the decimal point or the SET POINT character.

, Separates thousands with a comma (or with another character indicated by SET SEPARATOR)

A Restricts entry to alphabetic characters

L Restricts entry to T, t, F, f, Y, y, N, or n, and converts it to uppercase

N Restricts entry to letters, numbers and the underscore character.

X Allows any character

Y Restricts entry to Y, y, N, or n. Restricts display to Y and N

You may include *function* symbols in a template by preceding them with the @ symbol. If you combine template characters and function symbols in the same template, list function symbols first and separate them from the template characters with a space.

If the data is longer than the length of the *picture* string, it is truncated to match.

When displaying a calculated or morphed field, use a *picture* that represents the field's maximum size.

The **$** or ***** picture codes can be used interchangeably with **9** in a numeric picture string. Any ***** in the string overrides any **$** in the string.

For example;

When this template is used	The value 12.45	Will display as
99999.99		12.45
$$$$$.$$		$$$12.45
*****.**		***12.45
99999.99%		12.45%

popupEnable

Topic group Related topics

Whether an editor's popup menu is available.

Property of

Editor

Description

An Editor object has a popup menu that contains options to:

- Find and replace text
- Toggle its *wrap* and *evalTags* properties
- Show the Format toolbar

You may set the *popupEnable* property to *false* to prevent this menu from appearing when the user right-clicks the Editor object.

popupMenu

Topic group Related topics Example

Specifies a pop-up menu for a form.

Property of

Form, SubForm

Description

After creating the Popup object as a child of the form, assign a reference to that Popup object to the form's *popupMenu* property to have the popup appear when the user right clicks on the form. In this way, you may have more than one popup menu defined for a form and change the popup menu that appears as needed.

Note

You cannot name your popup *popupMenu*; that would conflict with the name of the existing property.

prefixEnable

Topic group Related topics

Determines whether the ampersand (&) character is interpreted as the accelerator key prefix.

Property of

Text, TextLabel

Description

When *prefixEnable* is *true*, the ampersand character is interpreted as the accelerator key prefix; the ampersand is not displayed, and the character that follows it is used as the accelerator key for the control that follows the Text or TextLabel object in the z-order. The accelerator key appears underlined.

Set *prefixEnable* to *false* to treat the ampersand as a normal character, so that it can be displayed within a Text or TextLabel control.

prevSibling

Topic group Related topics

The previous tree item with the same parent.

Property of

TreeItem

Description

The read-only *prevSibling* property contains an object reference to the previous tree item (up) that has the same parent. If the tree item is the first one, *prevSibling* is *null*.

print( )

Topic group Related topics

Prints a form as it appears on screen, or prints only the data from a completed form

Syntax

<oRef>.*print*([<dialog expL>[, <mode expL>]])

<oRef>

An object reference to the form you want to print.

<dialog expL>

An optional parameter that determines whether to display the standard print dialog. If omitted, the dialog is displayed by default. If the dialog is not displayed, the form is printed according to the settings of *_app.printer*.

<mode expL>

An optional parameter that selects which method to use when printing. This parameter defaults to *true* and can be passed via either of the following:

false Specifies that a screen image of the form will be printed

true Specifies that the form's data be printed for use in filling out a printed form

Property of

Form, SubForm

Description

Use the *print*( ) method to print a form on a selected printer. Executing the *print*( ) method opens the standard Print dialog box. If the user clicks OK, the current page of the form is printed on the selected printer.

When passing *true* as a second parameter:

You may set the *printable* property of individual controls to *false* to prevent them from printing.

printable

Topic group Related topics

Whether the component is printed when the form is printed.

Property of

Most form components.

Description

You may suppress the printing of individual components on the form by setting their *printable* property to *false*.

rangeMax

Topic group Related topics

Determines the upper limit for values in a component.

Property of

Progress, ScrollBar, Slider, SpinBox

Description

Use *rangeMax* in combination with *rangeMin* to specify a range restriction for values entered into a component. (*rangeMax* sets the upper limit and *rangeMin* sets the lower limit.) For example, an application that lets the user input a percentage would prevent the input of a value less than 0 or greater than 100. The same ranges would apply for a Progress component showing percent complete.

In a SpinBox component, if the value is too high, the value is set to *rangeMax*. SpinBox components allow both numbers and dates; the *rangeMax* must be the same data type. The Progress, Slider, and ScrollBar allow numbers only. *rangeMax* must be greater than *rangeMin*.

Note

Range restrictions in a SpinBox have effect only when the *rangeRequired* property is *true*.

rangeMin

Topic group Related topics

Determines the lower limit for values in a component.

Property of

Progress, ScrollBar, Slider, SpinBox

Description

Use *rangeMin* in combination with *rangeMax* to specify a range restriction for values entered into a component. (*rangeMin* sets the lower limit and *rangeMax* sets the upper limit.) For example, an application that lets the user input a percentage would prevent the input of a value less than 0 or greater than 100. The same ranges would apply for a Progress component showing percent complete.

In a SpinBox component, if the value is too low, the value is set to *rangeMin*. SpinBox components allow both numbers and dates; the *rangeMin* must be the same data type. The Progress, Slider, and ScrollBar allow numbers only. *rangeMin* must be less than *rangeMax*.

Note

Range restrictions in a SpinBox have effect only when the *rangeRequired* property is *true*.

rangeRequired

Topic group Related topics

Determines whether the range you specify with the *rangeMax* and *rangeMin* properties is enforced.

Property of

SpinBox

Description

Set *rangeRequired* to *true* to enforce a range limitation specified by the *rangeMax* and *rangeMin* properties. You may set *rangeRequired* to *false* to temporarily disable range checking.

When range checking is active, existing values are checked when they are displayed in the control. The spinbox also will not allow the entry of a number that is higher than *rangeMax* or lower than *rangeMin*. If the number—an existing number or new number—is out of range, the spinbox will change the number to the range limit; to *rangeMax* if the number is too large, or to *rangeMin* if the number is too small.

readModal( )

Topic group Related topics Example

Opens a form as a modal window and returns an optional value.

Syntax

<oRef>.*readModal*( )

<oRef>

The form to open.

Property of

Form

Description

Use the *readModal*( ) method to open a form as a modal window. A modal window has the following characteristics:

While the form is open, focus can't be transferred to other forms.

Execution of the routine that opened the form stops until the form is closed. When the form is closed, control transfers to the statement after the one that opened the form.

The form's *close*( ) method takes an optional parameter. If the form is closed with a parameter, the value of that parameter is returned by *readModal*( ). Otherwise *readModal*( ) returns *false*.

Many applications use modal forms as dialog boxes, which typically require users to take an action before the dialog box can be closed.

You can't open a form with the *readModal*( ) method when the MDI property is set to *true*.

To open a form as a modeless window, use the *open*( ) method.

reExecute( )

Topic group Related topics Example

Re-executes the report.

Syntax

<oRef>.*reExecute*( )

<oRef>

The ReportViewer object that contains the report to re-execute.

Property of

ReportViewer

Description

Call *reExecute*( ) to execute the report again with a new set of parameters. To render another page in the existing report, call the report's *render*( ) method through the ReportViewer object's *ref* property.

ref

Topic group Related topics Example

A reference to the Report object in a ReportViewer.

Property of

ReportViewer

Description

Use the ReportViewer object's *ref* property to access the report displayed.

refresh( )

Topic group Related topics

Redraws the form or grid.

Syntax

<oRef>.*refresh*( )

<oRef>

The object to refresh

Property of

Form, Grid, SubForm

Description

Use a form's *refresh*( ) method to update the controls on the form to reflect the current state of the data in the buffer when using tables in work areas. To update the data buffer, use the REFRESH command first. When using the data objects, call the rowset's *refreshControls*( ) method instead.

Call a grid's *refresh*( ) method to repaint the grid. The current row in the rowset becomes the top row in the grid, and all visible columns are repainted from the current row down.

refreshAlways

Topic group Related topics

Whether to update the form after all form-based navigation and updates.

Property of

Form, SubForm

Description

Set *refreshAlways* to *false* when performing extensive navigation or data processing during an event. When *refreshAlways* is *true*, *dBASE Plus* updates the form periodically during processing, which causes flicker and slows the process. When *refreshAlways* is *false*, the form is not updated until the event has completed.

You may force the update for the form during the event by calling *refresh*( ).

release( )

Topic group Related topics

Explicitly releases an object from memory.

Syntax

<oRef>.*release*( )

<oRef>

An object reference to the object you want to release.

Property of

All form objects; all report objects except Band and StreamFrame.

Description

dBASE Plus manages memory and resources used by objects automatically. When there are no more variables or properties that reference an object and that object is not visible onscreen, the

object is destroyed. Any components that are contained in the object, such as the components of a form, are also destroyed when the container is destroyed. Because of this automatic object management, there is usually no reason to call *release*( ).

release( ) explicitly releases an object from memory, returning *true* if successful. Any references that point to that object become invalid; attempting to use such a reference results in an error. If these references are tested with EMPTY( ), it returns *true*.

For example, you might want to get rid of a single component in a form. You could *release*( ) that component, but in most cases you could just as easily hide the component by setting its *visible* property to *false*.

releaseAllChildren( )

Topic group Related topics

Deletes all tree items in the tree.

Syntax

<oRef>.*releaseAllChildren*( )

<oRef>

The TreeView object you want to clear.

Property of

TreeView

Description

Call *releaseAllChildren*( ) to delete the entire contents of a tree view so that you can start over. To delete an individual tree item and it subtree, call the tree item's *release*( ) method.

right

Topic group Related topics

The position of the right edge of an object relative to its container.

Property of

Line

Description

The unit of measurement in a form or report is controlled by its *metric* property. The default metric for forms is characters, and for reports it's twips.

rowSelect

Topic group Related topics

Whether the entire row is visually selected.

Property of

Grid

Description

Set *rowSelect* to *true* to create the visual effect of highlighting the entire row in the grid.

rowset

Topic group Related topics Example

The form's primary rowset.

Property of

Form, SubForm

Description

For forms that use data objects, the *rowset* property identifies the form's primary rowset.

If the form uses only one query, then the Form designer assigns that query's rowset as the primary rowset. If the form uses a data module, the Form designer assigns the data module's primary rowset as the form's primary rowset.

The primary rowset is where navigation and other actions from the default menu and toolbar take place. Navigation in the primary rowset causes the form's *canNavigate* and *onNavigate* events to fire.

saveRecord( )

Topic group Related topics

Saves changes to the current record in the currently active table.

Syntax

<oRef>.*saveRecord*( )

<oRef>

An object reference to the form.

Property of

Form, SubForm

Description

Use *saveRecord*( ) for form-based data handling with tables in work areas. When using the data objects, *saveRecord*( ) has no effect; use the rowset's *save*( ) method instead.

Form-based data buffering lets you manage the editing of existing records and the appending of new records. Editing changes to the current record are not written to the table until there is navigation off the record, or until *saveRecord*( ) is called. Each work area has its own separate edit buffer. For example, if you have two controls *dataLink*ed to two different work areas, and you change both controls, you must call *saveRecord*( ) while each work area is selected to commit the changes.

To append a new record, call *beginAppend*( ). This empties the record buffer for the currently selected work area. Calling *saveRecord*( ) writes the new record to the table, leaving you at the newly added record. Calling *beginAppend*( ) instead of *saveRecord*( ) will write the new record and empty the buffer again so you can add another record.

When appending records with *beginAppend*( ) the new record will not be saved when you call *saveRecord*( ) unless there have been changes to the record; the blank new record is

abandoned. This prevents the saving of blank records in the table. (If you want to create blank records, use APPEND BLANK). You can check there have been changes by calling *isRecordChanged*( ). If *isRecordChanged*( ) returns *true*, you should validate the record with form-level or row-level validation before writing it to the table.

To abandon the changed or new record instead of saving it, call *abandonRecord*( ).

scaleFontBold

Topic group Related topics

Whether the base font used for the Chars *metric* of a form is boldface.

Property of

Form, Report

Description

The Chars *metric* of a form is based on the average height and width for characters in a specific base font. The base font is set through the form's *scaleFontBold*, *scaleFontName*, and *scaleFontSize* properties.

The *scaleFontBold* propertydetermines whether the base font is boldface. Boldface fonts are wider than non-boldface fonts.

scaleFontName

Topic group Related topics

The base font typeface used for the Chars *metric* of a form.

Property of

Form, Report

Description

The Chars *metric* of a form is based on the average height and width for characters in a specific base font. The base font is set through the form's *scaleFontBold*, *scaleFontName*, and *scaleFontSize* properties.

scaleFontName is the name of the base font typeface.

scaleFontSize

Topic group Related topics

The point size of the base font used for the Chars *metric* of a form.

Property of

Form, Report

Description

The Chars *metric* of a form is based on the average height and width for characters in a specific base font. The base font is set through the form's *scaleFontBold*, *scaleFontName*, and *scaleFontSize* properties.

The *scaleFontSize* property designates the size of the base font, in "points".

scroll( )

Topic group Related topics

Gives the appearance of scrolling the client area, of a form or subform, by moving the form's border window over the form's client area.

Syntax

<oRef>.*scroll*(<horizontal expN>,<vertical expN>)

<oRef>

An object reference to the Form or SubForm.

<horizontal expN>

Horizontal position of top left corner of the form’s client area

<vertical expN>

Vertical position of top left corner of the form’s client area

Property of

Form, SubForm

Description

The *scroll*( ) method allows you to programatically "scroll" the client area of a form or subform. Relative to the form's initial position at (0,0), the form will move over the client area to a position designated by the values of <horizontal expN> and <vertical expN>.

Calling form.scroll(10, 10) while form.metric is set to 0 - Chars, moves the form's border window to a position 10 chars to the right and 10 chars down from its initial position of 0, 0. Any controls or data on the form will appear to be scrolled up and to the left of their initial positions.

By subsequently calling form.scroll(20, 20), the form's border window will be scrolled to a position 20 chars down and 20 chars to the right of its initial position, 0,0. Again the form's contents will appear to move up and to the left.

If you next call form.scroll(5, 5), the form's border window will be scrolled to a position 5 chars down and 5 chars to the right of its initial position, 0, 0. However, since the previous position (20,20) was further down and to the right, the form's contents will appear to be scrolled down and to the right relative to their previous positions.

Note that the coordinates specified are always relative to the initial position of the form, not the form's most recent position.

To use the *scroll*( ) method, a form or subform's scrollbar must be visible and enabled. This can be done by;

Setting the *scrollBar* property to 1 (On).
or
Setting the *scrollBar* property to 2 (Auto), and sizing the form or subform so it's client area, or contents, occupy a larger area than the form itself. This displays and enables the scrollbars.

scrollBar

Topic group Related topics

Determines when an object has a scroll bar.

Property of

Browse, Editor, Form, ReportViewer, SubForm

Description

The *scrollBar* property determines when and if a control displays a scrollbar. It may have any of four settings:

Value	Description
0 (Off)	The object never has scroll bars.
1 (On)	The object always has scroll bars.
2 (Auto)	Displays scroll bars only when needed.
3 (Disabled)	The scroll bars are visible but not usable.

scrollHOffset

Topic group Related topics

Contains the current position of the horizontal scrollbar in units matching the form or subform's current metric property.

Property of

Form, Subform

Description

The value in the *scrollHOffset* property is relative to the left edge of the form. On a form with no scrollbars, or with its horizontal scrollbar in its unscrolled position, the value of the *scrollHOffset* property is zero.

As the horizontal scrollbar is moved to the right, the value of the *scrollHOffset* property increases.

As the horizontal scrollbar is moved to the left, the value of the *scrollHOffset* property decrease

scrollVOffset

Topic group Related topics

Contains the current position of the vertical scrollbar in units matching the form or subform's current metric property.

Property of

Form, Subform

Description

The value in the *scrollVOffset* property is relative to the top of the form. On a form with no scrollbars, or with its vertical scrollbar in its unscrolled position, the value of the *scrollVOffset* property is zero.

As the vertical scrollbar is moved downward, *scrollVOffset* increases.

As the vertical scrollbar is moved upward, *scrollVOffset* decreases.

select( )

Topic group Related topics

Makes the tree item the selected item in the tree.

Syntax

<oRef>.*select*( )

<oRef>

An object reference to the tree item you want to select.

Property of

TreeItem

Description

Use *select*( ) to programmatically select a tree item. The tree view is scrolled and expanded if necessary to display the selected tree item.

selectAll

Topic group Related topics

Determines if the value contained in a control initially appears selected (highlighted) when tabbing to the control.

Property of

Combobox, Entryfield, SpinBox

Description

Set selectAll to true to give the user a shortcut for deleting or replacing the initial value in an entry field, a spin box, or a combobox. The entire value is highlighted when the user tabs to the control. The first character the user enters overwrites the value. Pressing Del or Backspace deletes the value. Pressing a direction key (such as Home or End) removes the highlight without erasing the value.

Clicking a control ignores selectAll; the cursor goes to the position clicked, and nothing is highlighted.

comboBox ...

- The default for selectAll is True.
- Setting selectAll to False prevents the entire combobox value from being highlighted when the combobox is given focus or when scrolling through option strings in the dropdown list.

selected

Topic group Related topics

The currently selected tree item.

Property of

TreeView

Description

The *selected* property contains an object reference to the currently-selected tree item in the tree view, the one that has focus. If no item is selected, *selected* contains *null*.

selected is usually used in TreeView event handlers, which fire for all items in the tree.

selected is read-only. To programmatically select an item, call the item's *select*( ) method.

selected( )

Topic group Related topics Example

Returns the currently selected item(s) in an object, or checks if a specified item is selected.

Syntax

<oRef>.*selected*([<item expN>])

<oRef>

A reference to the object you want to check.

<item expN>

The item number to check. If omitted, the currently selected item(s) are returned. This parameter is allowed only for the ListBox control; the Grid method takes no parameters.

Property of

Grid, ListBox

Description

Calling *selected*( ) with no parameters returns the control's currently selected item or items. If a listbox's *multiple* property is *false*, *selected*( ) returns the currently selected item in the listbox. If *multiple* is *true*, *selected*( ) returns an array containing the currently selected items, one element per selection.

For a grid, *selected*( ) returns bookmarks to the row or rows (in an array) that are selected when *rowSelect* or *multiSelect* are *true*.

For a ListBox, *selected*( ) also returns bookmarks when the datasource is a field object.

As with any Array object, check its *size* property to determine how many items have been selected. The items in the *selected*( ) array are listed in the order they were selected.

If you specify <item expN>, *selected*( ) will return the prompt string for that numbered item if the item is selected, or an empty string if the item is not selected.

The ListBox object's *value* property contains the value of the item that currently has focus, whether it's selected or not.

selectedImage

Topic group Related topics

Image displayed between checkbox and text label when a tree item has focus.

Property of

TreeItem, TreeView

Description

The tree view may display images to the left of the text label of each tree item. If the tree has checkboxes, the image is displayed between the checkbox and the text label.

The *selectedImage* property of the TreeView object specifies the default icon image for all tree items to display when that tree item has focus. You may designate specific icons for each TreeItem object to override the default. Use the *image* property to specify icons for when the tree item does not have focus. If any individual item in the tree has its *image* or *selectedImage* property set, space is left in all tree items for an icon, even if they don't have one.

The *selectedImage* property is a string that can take one of two forms:

RESOURCE <resource id> <dll name>
specifies an icon resource and the DLL file that holds it.

FILENAME <filename>
specifies an ICO icon file.

serverName

Topic group Related topics

Identifies the server application that is invoked when the user double-clicks an OLE viewer object.

Property of

OLE

Description

serverName is a read-only property that contains the name of the OLE server for the current contents of the OLE control. You may use *serverName* to anticipate which server application is activated if the user double-clicks the current OLE viewer object.

setAsFirstVisible( )

Topic group Related topics

Makes the tree item visible as the first (topmost) in the tree view.

Syntax

<oRef>.*setAsFirstVisible*( )

<oRef>

An object reference to the tree item you want to display.

Property of

TreeItem

Description

Use *setAsFirstVisible*( ) when you want to make sure that a tree item is visible in the tree view as close to the top of the tree view area as possible. The tree is expanded and scrolled if necessary to make the item visible.

If the tree item is close to the bottom, the tree is scrolled as high as possible.

setFocus( )

Topic group Related topics

Sets focus to a component.

Syntax

<oRef>.*setFocus*( )

<oRef>

A reference to the object to receive focus.

Property of

Form and all form components that get focus

Description

Calling a component's *setFocus*( ) method moves the focus to that component (unless it already has focus). When you call a form's *setFocus*( ) method, the component on the form that last had focus gets the focus.

setTic( )

Topic group Related topics

Manually sets a tic mark in a Slider object.

Syntax

<oRef>.*setTic*(<expN>)

<oRef>

The Slider object whose tic mark to set.

<expN>

The location of the tic mark.

Property of

Slider

Description

To manually set tic marks in a slider, set the slider's *tics* property to Manual. Call *clearTics*( ) to clear any previously set tic marks. Then call *setTic*( ) with the location of each tic mark.

The <expN> should be between the *rangeMin* and *rangeMax* of the Slider control.

setTicFrequency( )

Topic group Related topics

Sets the tic mark interval for automatic tic marks in a Slider object.

Syntax

<oRef>.*setTicFrequency*(<expN>)

<oRef>

The Slider object whose tic mark to set.

<expN>

The frequency of the tic marks.

Property of

Slider

Description

For automatic tic marks, set the slider's *tics* property to Auto. The default interval is 1. Call *setTicFrequency*( ) to use another interval value.

shapeStyle

Topic group Related topics

Determines the shape of a Shape object.

Property of

Shape

Description

Use *shapeStyle* to specify a shape for a Shape object. The shapes you can specify are as follows:

Value	Shape
0	Rectangle with rounded corners
1	Rectangle
2	Ellipse
3	Circle
4	Square with rounded corners
5	Square

showFormatBar( )

Topic group Related topics

Displays or hides the Format toolbar.

Syntax

<oRef>.*showFormatBar*(<expL>)

<oRef>

A Form object.

<expL>

true to show the toolbar, *false* to hide the toolbar.

Property of

Form, SubForm

Description

You may show or hide the Format toolbar when the form has focus by calling *showFormatBar*( ).

showMemoEditor( )

Topic group Related topics

Displays or hides the memo editor control for an entryfield.

Syntax

<oRef>.*showMemoEditor*(<expL>)

<oRef>

The Entryfield that's *dataLink*ed to the memo field.

<expL>

true to show the editor, *false* to hide the editor.

Property of

Entryfield

Description

When an entryfield control is *dataLink*ed to a memo field, double-clicking the control opens a memo editor in a form. You may show or hide this editor by calling *showMemoEditor*( ).

showSelAlways

Topic group Related topics

Whether to highlight the selected item in the tree even when the tree view does not have focus.

Property of

TreeView

Description

When *showSelAlways* is *true*, the tree view highlights the selected item even when the tree view does not have focus. This highlight is different than when the tree view does have focus, but still visually indicates the selected item.

When *showSelAlways* is *false*, no item is highlighted when the tree view does not have focus. Even though no item is highlighted, the tree view's *selected* property still points to the item that had focus last.

showSpeedTip

Topic group Related topics

Determines if tool tips defined for a control appear.

Property of

Form, SubForm

Description

Set *showSpeedTip* to false to suppress the display of all tool tips defined in the controls' *speedTip* property.

showTaskBarButton

Topic group Related topics Example

Determines if a button will be created on the Windows Taskbar for the form. For a non-mdi forms only.

Property of

Form

Default

True

Description

Set the *showTaskBarButton* property to *true* to display a button for the form on the Windows Taskbar. However, a Windows Taskbar button will not display if the form's *smallTitle* property is also set to *true*.

When a form's *showTaskBarButton* property is set to *false*, calling the form's *open*( ) method will cause its *hWndParent* property to be set to the *hWnd* property of a hidden parent window.

When the *showTaskBarButton* property is set to *true*, the *hWndParent* remains set at 0.

Switching between dBASE Plus, or a dBASE Application, and other Windows programs

Set the *showTaskBarButton* property to *false* before opening a form, via its *readModal*( ) method, to ensure that a modal form always stays on top when switching between dBASE Plus, or a dBASE Application, and other Windows programs.

When running a non-modal and non-mdi form, however, you must also set the form's *hWndParent* property to the appropriate parent hWnd (ex. _app.frameWin.hWnd), during the form's *open*( ) method, to ensure the form will always stay on top when switching between these programs.

sizeable

Topic group Related topics

Determines if the user can resize a form when it's not MDI.

Property of

Form, SubForm

Description

Set *sizeable* to *false* to prevent the form from being resized. You must set *sizeable* before you open the form.

sizeable has no effect unless the form's *mdi* property is *false*; if it is *true*, the form follows the MDI specification and is always resizeable.

smallTitle

Topic group Related topics

Determines if the form has the small palette-style title bar when it's not MDI.

Property of

Form, SubForm

Description

Set the *smalltitle* property to *true* to make the form look like a palette window, with the smaller title bar and no minimize, maximize, taskbar entry, or system menu icons. You must set *smallTitle* before you open the form.

The *smallTitle* property has no effect unless the form's *mdi* property is *false*. When the *mdi* property is *true*, the form follows the MDI specification and has a normal-sized title bar.

To enable a *showTaskBarButton* property to display a Windows Taskbar button, set the *showTaskBarButton* to *true* and the *smallTitle* properties to *false*.

sortChildren( )

Topic group

Sorts child tree items.

Syntax

<oRef>.*sortChildren*( )

<oRef>

The tree object whose children to sort.

Property of

TreeItem, TreeView

Description

sortChildren( ) sorts the child tree items of a tree view or tree item according to the *text* labels of the tree items. The sort is not case-sensitive and goes one level deep only.

sorted

Topic group Related topics

Determines whether the prompts in a listbox or a combobox are listed in sorted order or in natural order.

Property of

ComboBox, ListBox

Description

Set *sorted* to *true* when you want the prompts in a list box or a combo box to appear in sorted order (alphabetically, numerically, or chronologically). For example, a list of names is more accessible if it is sorted alphabetically.

The natural order of a list box or a combo box depends on the order in which the prompts are generated. For example, when you specify "FILE *.*" for the *dataSource* property of a list box, the prompts consist of the file names in the default directory. The prompts are created in the

order in which the files are listed in the directory, so they are not necessarily arranged alphabetically when you set *sorted* to *false*.

sorted has no effect when the *dataSource* property of the list box or combo box specifies "FIELD" followed by a field name. In this case, the order of prompts in the list box or combo box depends on the record sequence in the table containing the specified field.

speedBar [PushButton]

Topic group Related topics

Determines whether a pushbutton behaves like a toolbar button or a standard pushbutton.

Property of

PushButton

Description

Set *speedBar* to *true* when you want a pushbutton to behave like a toolbar button. A toolbar button is not included in the tab order of a form, so you can't get to it by pressing Tab or Shift+Tab; and when clicked, it does not receive focus.

For example, navigation controls on a form usually have their *speedBar* property set to *true*. When you navigate from row to row, the control that has focus, typically one *dataLink*ed to a field, never loses focus.

Note

When a PushButton's *speedBar* property is set to true, and therefore the PushButton cannot get focus, the form's *activeControl* property does not show the PushButton as being the current activeControl.

speedTip

Topic group Related topics

Specifies the tooltip text that appears when the mouse remains on a control for more than one second.

Property of

ActiveX, Browse, CheckBox, ComboBox, Container, Editor, Entryfield, Grid, Image, ListBox, NoteBook, OLE, PaintBox, Progress, PushButton, RadioButton, Rectangle, ScrollBar, Slider, SpinBox, TabBox, Text, TextLabel, ToolButton, TreeView

Description

Use the *speedTip* property to create a tool tip that appears when the mouse rests on a control. Usually this message gives the user a clue as to the function of the control. To suppress the display of these tool tips, set the *showSpeedTip* property of the form to *false*.

A speedTip's text will appear whether or not the relevant object has focus (In *dBASE Plus*, an object does not get focus as the cursor passes over).

A speedTip's text will not appear unless the relevant object is;

enabled (it's *enabled* property set to true)

and

visible (it's *visible* property set to true)

Whether or not a PushButton's speedTip text appears does not depend on the setting of it's *speedBar* property.

spinOnly

Topic group Related topics

Determines if users can enter a value in the text box portion of a spin box.

Property of

SpinBox

Description

A spinbox lets users enter values in a text box or select values by clicking the arrows on the right edge of the spinbox. When you set the *spinOnly* property to *false*, the text box is enabled. When you set the *spinOnly* property to *true* the text box is disabled, restricting input to a predefined *step* value.

startSelection

Topic group Related topics

The low end of the selection range in a Slider object.

Property of

Slider

Description

startSelection contains the low value in the selection range. It should be equal to or less than *endSelection*, and between the *rangeMin* and *rangeMax* values of the slider.

The selection is not displayed unless the slider's *enableSelection* property is *true*.

statusMessage

Topic group Related topics

The message to display on the status bar while an object has focus.

Property of

Form and all form components that can receive focus

Description

Use *statusMessage* to provide instructions to the user when the user selects an object. If you set the *statusMessage* of the form, that message is displayed in the status bar when a component's *statusMessage* is blank.

step

Topic group Related topics

Determines how the amount added or subtracted by clicking an arrow in a spinbox.

Property of

SpinBox

Description

Use *step* to control the rate at which a user can increase or decrease a numeric or date value. For example, a program that expresses large dollar values only in increments of $500.00 would give a spinbox a *step* value of 500.

streamChildren( )

Topic group Related topics

Saves child TreeItem objects and properties to a text file.

Syntax

<oRef>.*streamChildren*(<filename expC>)

<oRef>

The TreeView parent object of the child TreeItems.

<filename expC>

The name of the file to contain the child TreeItem objects and properties.

Property of

TreeView

Description

Use *streamChildren*( ) to save the child TreeItems of a TreeView object to a text file. The filename may be any valid filename. If the file exists, it will be overwritten; if not, it will be created.

streamChildren( ) streams the class definition and properties (except for TreeItem method overrides) of all of the TreeItems in the TreeView. It does *not* stream the class definition for the parent TreeView, thereby enabling you to attach the streamed TreeItems to a different TreeView object using that TreeView's *loadChildren*( ) method.

Note:

streamChildren( ) will overwrite any existing filename without warning, even with SAFETY ON. Always first check for the existence of the new filename, or use the FUNIQUE( ) function to create a unique filename.

style

Topic group Related topics

Specifies which parts of a combobox are usable and which parts are displayed automatically.

Property of

ComboBox

Description

Use *style* to determine how the user selects values in a combobox.

The user selects a value from a combobox by entering initial characters in an entryfield or by selecting the value directly from the prompt list. The *style* property determines whether the entryfield accepts input, and how the prompt list is displayed.

You can give Style one of three values:

Style value	Description
0 (Simple)	The prompt list does not drop down, and there is no arrow button. The combobox height property determines how much of the prompt list is displayed (the default is no prompts, only the entryfield is displayed). If the combobox contains more prompts than can be displayed in the visible list portion, a scrollbar will appear. The user can select from the list or type in the entryfield.
1 (DropDown)	The user has to click the arrow to display the drop-down list. The user can either type in the entryfield, or select from the list.
2 (DropDownList)	The user has to click the arrow to display the drop-down list. The entryfield does not accept input; the user must choose from the list.

Set autoDrop to *true* to make the prompt list drop automatically when a style 1 or 2 combobox gets focus.

Pressing Alt+DownArrow when the combobox has focus also displays the drop-down list.

sysMenu

Topic group Related topics

Determines if a form has a control menu and close icon when it's not MDI.

Property of

Form, SubForm

Description

Set the *sysMenu* property to *false* to hide the control menu and close icon on a form. You must set the *sysMenu* property before you open the form.

The *sysMenu* property has no effect unless the form's *mdi* property is *false*; if it is *true*, the form follows the MDI specification and always has a control menu and close icon.

systemTheme

Topic group

Determines whether a form object is displayed using the current XP or Vista Visual Style or the Classic Windows style.

Property of

Most form objects.

Default

true

Description

Set the systemTheme property to true (the default) to display the object using the current Windows XP or Windows Vista Visual Style.

Set the systemTheme property to false to display the object using the Classic style.

When the systemTheme property is set to true, the following conditions must also be met in order to display the object using the Windows XP or Windows Vista Visual Style:

dBASE Plus or a dBASE Plus runtime application must be running on Windows XP, or newer version of Windows.

A Windows XP or Windows Vista manifest file must be in use for dBASE Plus or a dBASE Plus runtime application. Within this manifest file, the common controls must be set to the correct version in order to see the most current OS style.

tabStop

Topic group Related topics

Determines if the user can select an object by pressing Tab or Shift+Tab.

Property of

All form components that can receive focus

Description

Set the *tabStop* property to *false* when you want to remove an object from the tab order of the parent form. For example, a TabBox object to select pages on a form is often not put in the tab order because it's not a data entry control. (For data entry purposes, you could put a button at the end of the tab order to move the user to the next page.)

If you have a PushButton component that you don't want in the tab order, you may not want it to receive focus either. If that's the case, set its *speedBar* property to *true* instead, which prevents both tabbing and focus.

text

Topic group Related topics Example

The non-editable text that appears in a component.

Property of

Browse, CheckBox, Form, Menu, PushButton, RadioButton, Rectangle, SubForm, Text, TextLabel, ToolBar,TreeItem

Description

The *text* of a CheckBox or RadioButton object is the descriptive text that appears beside the actual check box or RadioButton. The *text* of a TreeItem object is the item's editable text label. The *text* of a PushButton object is the text that appears on the button face, and the *text* of a Menu object is the menu prompt.

The *text* of a Rectangle object is the caption text that appears at the top of the control. The *text* of a Form object is the text that appears in the form's title bar. The *text* of a floating ToolBar is the text that appears in it's title bar. The *text* property has no effect in a Browse object; the property exists for compatibility only.

The *text* of a Text or TextLabel object is the content of the object. For a Text object, this is the actual HTML text that is displayed in the form or report.

You may assign any of the following types of data to the *text* property of a Text component:

Boolean
Numeric
Integer

Character
Object
Null
DateTime
Codeblock

If you assign a codeblock to the *text* property, it must return a value. Use either an expression codeblock or a statement codeblock that uses RETURN to return a value. The codeblock is evaluated whenever it is rendered.

Creating Accelerator Keys

Use a pick character to create an accelerator key to let the user select a menu item or simulate clicking a CheckBox, RadioButton, or PushButton by pressing Alt and the pick character. To designate a character as a pick character, precede it with an ampersand (&) in the object's *text* property. The pick character is underlined when the object is displayed. A pick character may also be used for the *text* of a Text, TextLabel or Rectangle object; pressing the key combination gives focus to the first control that follows the object in the z-order that can receive focus.

Note

When a PushButton's *speedBar* property is set to "true", accelerator keys are ignored.

textLeft

Topic group Related topics

Whether the component's text displays to the left or the right of the graphic element.

Property of

CheckBox, PushButton, RadioButton

Default

false

Description

The CheckBox, PushButton, and RadioButton objects combine a text label and a graphic element.

Set *textLeft* to *true* if you want the text to display on the left side of the control. By default, *textLeft* is *false*, so the text displays on the right.

The PushButton uses the *upBitmap*, *downBitmap*, *focusBitmap*, and *disabledBitmap* properties to display a bitmap on the button face and, for a pushButton, the *textLeft* property.can be overridden by the *bitmapAlignment* property.

tics

Topic group Related topics

How to display the tic marks in a Slider object.

Property of

Slider

Description

tics is an enumerated property that can be one of the following values:

Value	Description
0	Auto
1	Manual
2	None

If *tics* is Auto, set the tic mark interval with *setTicFrequency*( ). For Manual tics, use the *setTic*( ) method. Use the *ticsPos* property to set where the tic marks are displayed.

ticsPos

Topic group Related topics

Where to display the tic marks in a Slider object.

Property of

Slider

Description

ticsPos is an enumerated property that can be one of the following values:

Value	Description
0	Both
1	Bottom Right
2	Top Left

Tic marks are displayed if the *tics* property is not set to None. If the slider is vertical, the tic marks are displayed on the right or left side of the slider, or both. If the slider is horizontal, the tic marks are displayed on the top or bottom, or both.

Make sure the Slider object is large enough to display the tic marks.

toggle

Topic group Related topics

Determines if a button acts as a two-state toggle.

Property of

PushButton

Description

Set *toggle* to *true* to have a PushButton object behave like a two-state toggle button that stays down when clicked. Clicking the button again makes it pop back up.

To check the state of a toggle button, check its *value* property.

toolTips

Topic group Related topics

Whether to display text labels as tooltips if they are too long to display fully in the tree view area as the mouse passes over them.

Property of

TreeView

Description

When *toolTips* is *true*, tree item text labels are displayed as tooltips if necessary as the mouse passes over them. The tooltip displays directly over the obscured text label, in the exact position where the text label should appear.

In general, this allows the user to see the entire text label without having to scroll the tree view back and forth horizontally. However, some portion of the tree item must be visible; if the tree item is completely outside the tree view area, the item will not appear simply by pointing the mouse where the item would be (because the mouse is outside the bounds of the TreeView object).

top

Topic group Related topics

The position of the top edge of an object relative to its container.

Property of

Form, SubForm and all form contained objects.

Description

The unit of measurement in a form or report is controlled by its *metric* property. The default metric for forms is characters, and for reports it's twips.

The container for an MDI form is the main *dBASE Plus* application window, also known as the MDI frame window, below the menu and any toolbars docked on the top of the window. For a non-MDI form, the container is the screen.

topMost

Topic group Related topics

Specifies whether a form displays on top of all other forms when it's not MDI.

Property of

Form, SubForm

Description

Set a form's *topMost* property to *true* to make the form stay in the foreground while focus transfers to other windows. If more than one open form has *topMost* set to *true*, those windows behave normally in relation to each other, while always staying on top of all other windows.

topMost has an effect only when the *mdi* property is *false*.

trackSelect

Topic group Related topics

Whether to highlight and underline tree items as the mouse passes over them.

Property of

TreeView

Description

Set *trackSelect* to *true* to give the user an extra visual indication of which tree item the mouse is currently over.

transparent

Topic group Related topics

Specifies whether an object's background matches the background color, or image, of its container.

Property of

CheckBox, Container, ListBox, PaintBox, RadioButton, Rectangle, Text, TextLabel

Description

By setting an object's *transparent* property to *true*, its background takes on the background color, or image, of its container, making the background appear to be transparent. Note that the background is not actually transparent; if the control overlaps another control, you will still see the background of the container, not the portion of the control that has been overlapped.

uncheckedImage

Topic group Related topics

The image to display when a tree item is not checked instead of an empty check box.

Property of

TreeView

Description

Use *uncheckedImage* to display a specific icon instead of the standard empty checkbox for the tree items in the tree that are not checked. *checkedImage* optionally specifies the icon to display for tree items that are checked. The tree must have its *checked* property set to *true* to enable checking; each tree item has a *checked* property that reflects whether the item is checked.

The *uncheckedImage* property is a string that can take one of two forms:

RESOURCE <resource id> <dll name>
specifies an icon resource and the DLL file that holds it.
FILENAME <filename>
specifies an ICO icon file.

undo( )

Topic group Related topics

Reverses the effects of the last Cut or Paste action.

Syntax

<oRef>.*undo*( )

<oRef>

An object reference to the control you want to restore.

Property of

Browse, ComboBox, Editor, Entryfield, SpinBox

Description

Use *undo*( ) when the user wants reverse the effects of the last Copy, Cut or Paste action.

If you have assigned a menubar to the form, you can use a menu item assigned to the menubar's *editUndoMenu* property instead of using the *undo*( ) method of individual objects on the form.

upBitmap

Topic group Related topics

Specifies the graphic image to display in a pushbutton when it isn't selected.

Property of

PushButton

Description

Use *upBitmap* to give visual confirmation that a pushbutton is enabled and the user is not clicking it.

The *upBitmap* setting can take one of two forms:

RESOURCE <resource id> <dll name>
specifies a bitmap resource and the DLL file that holds it.

FILENAME <filename>
specifies a bitmap file. See class Image for a list of bitmap formats supported by *dBASE Plus*.

When you specify a character string for the pushbutton with *text* and an image with *upBitmap*, the image is displayed with the character string.

useTablePopup

Topic group Related topics

Specifies whether to use the default table navigation popup when no popup has been assigned to a form.

Property of

Form, SubForm

Description

Set *useTablePopup* to *true* to have the default table navigation popup displayed when the user right-clicks the form and no popup has been assigned to the form's *popupMenu* property.

If a popup is assigned to *popupMenu*, that popup is always used.

valid

Topic group Related topics Example

Event fired when attempting to leave a control; return value determines if focus can be removed.

Parameters

none

Property of

ColumnEditor, Editor, Entryfield, SpinBox

Description

Use *valid* to validate data. *valid* fires when attempting to leave the control only when data has been changed, unless *validRequired* is *true*; then *valid* always fires.

The *valid* event handler must return *true* or *false*. If it returns *true*, operation continues normally. If it returns *false*, the *validErrorMsg* is displayed in a dialog box and the focus remains on the control.

valid does not fire if the user never visits the control, even if *validRequired* is *true*. Therefore, unless the control is the first or only control on the form that gets focus, you should always use form-level or row-level validation in addition to control-level or field-level validation.

To enforce a simple numeric range in a SpinBox control, use *rangeMax* and *rangeMin* instead of *valid*.

To perform an action when a control loses focus, use *onLostFocus*.

validErrorMsg

Topic group Related topics

Specifies the message to display when the *valid* event handler returns *false*.

Property of

Entryfield, SpinBox

Description

When validating data with *valid*, set the *validErrorMsg* property of the control to a more specific error message than the default, "Invalid input". You may set a static message for each specific control, for example "Bad account number"; or you may dynamically set the *validErrorMsg* from within the *valid* event with specifics about the particular error, for example "Account number requires six digits" or "Account expired".

validRequired

Topic group Related topics

Determines if the *valid* event fires even if the data is not changed.

Property of

Entryfield, SpinBox

Description

Set *validRequired* to *true* to validate existing data as well as new data. For *validRequired* to take effect, you must assign a *valid* event handler.

You typically set *validRequired* to *true* when you change a validation condition and need to verify and update existing data. For example, a business might add a digit to its account numbers and change the *valid* event handler of an entry field to require the new digit. If the *validRequired* property is set to *true*, *dBASE Plus* also detects any existing account numbers that lack the digit and forces the user to make appropriate changes.

valid does not fire if the user never visits the control, even if *validRequired* is *true*. Therefore, unless the control is the first or only control on the form that gets focus, you should always use form-level or row-level validation in addition to control-level or field-level validation.

value

Topic group Related topics

The component's current value.

Property of

CheckBox, ColumnEditor, ComboBox, Editor, Entryfield, ListBox, Progress, PushButton, RadioButton, ScrollBar, Slider, SpinBox

Description

A component's *value* property reflects its value, which is

- The value that is displayed in a ColumnEditor, Entryfield, Editor, SpinBox, or ComboBox component
- *true* if a CheckBox component is checked; *false* if it's not checked
- *true* if a RadioButton component is the one in its group that is selected; *false* if it's not selected
- The item that has focus in a ListBox component
- The interpolated number for the current position in a Slider, Progress, or ScrollBar object.
- *true* if a *toggle* PushButton is down; *false* if it's up

Both field and component objects have a *value* property. (Fields in a table open in a work area do not have any properties, but they have a value; the concept is the same). When they are *dataLink*ed, changes in one object's *value* property are echoed in the other. The form component's *value* property reflects the value displayed in the component at any given moment. If the component's value is changed, it is copied into the field.

The *value* property for all fields in a rowset is set when you first open a query, and updated as you navigate from row to row. The *value* properties for components *dataLink*ed to those fields are also updated at the same time, unless the rowset's *notifyControls* property is set to *false*. You can also force the components to be updated by calling the rowset's *refreshControls*( ) method, which is useful if you have set a field's *value* property through code.

When reading or writing values to *dataLink*ed components, you can use the *value* property of either the visual component or the field object; there's no difference, although you should be consistent. You may choose to program the visual interface, if the underlying data is more likely to change; or you might choose to work with the data objects, so you don't have to worry about the names of the form components and whether they're correctly *dataLink*ed. In general, it's easier and more portable for data object events to access the fields, so you're more likely to assign to the *value* properties of the fields.

When the *multiple* property of a ListBox component is set to *true*, the "item that has focus", and subsequently determines the value of the *value* property, is the most recently selected, or unselected, item. Do not assume an item has focus because it is the last highlighted item on a

list or because it was the most recently highlighted item. For example, suppose you have a ListBox containing options 1 through 4 which are then selected in the order 1, 4, and 2. Although the last highlighted item on the list is selection 4, the *value* property would be determined by the value of option 2, the last selected option. In the event that option 1 is then unselected, it would receive focus and determine the value of the *value* property even though it is not one of the highlighted items.

vertical

Topic group Related topics

Determines whether a Slider or ScrollBar object is vertical or horizontal.

Property of

ScrollBar, Slider

Description

Slider and ScrollBar objects may be horizontal or vertical. If *vertical* is *true* it's vertical; if *vertical* is *false*, it's horizontal.

Changing the *vertical* property does not resize the object. For example, if you have a long horizontal scrollbar, setting *vertical* to *true* results in a short, fat vertical scrollbar.

view

Topic group Related topics Example

Specifies the name of a .QBE query or a table on which a form is based.

Property of

Form, SubForm

Description

Use *view* for form-based data handling with tables in work areas. When using the data objects, do not use *view*.

view determines which tables are automatically opened whenever the form is opened. You may specify a single table, or a .QBE query file. If you specify a single table, *dBASE Plus* internally issues CLOSE DATABASES to close all tables open in work areas (in the current workset) before opening the specified table in work area 1, in its natural or primary key order.

A .QBE file is a program file that is supposed to open one or more tables in a specific index order, and contains the appropriate Xbase DML commands such as SET RELATION and SET SKIP to create a multi-table view. In .QBE files generated by earlier versions of dBASE, the first command in the file is CLOSE DATABASES, so using a generated .QBE also closes all open tables.

The specified table is opened (or the .QBE is executed) immediately when the *view* property is assigned.

Instead of using the *view* property, you may open the necessary tables yourself. All tables containing fields that are *dataLink*ed to controls on the form must be open when the form is instantiated; otherwise the *dataLink* properties will fail (because the specified fields cannot be found), causing an error.

If one form opens another form that is supposed to use the current record in the first form, you don't want to set the *view* property in the second form, because instantiating that second form would close the tables used by the first form. This is a common situation where a form would use the current view, and not have anything assigned to its *view* property.

If a form does not have its own *view*, you may assign a *designView* property to the form so that the necessary tables are opened when you edit the form in the Form designer. The *designView* property has no effect when you actually run the form.

visible

Topic group Related topics

Specifies whether a component is visible.

Property of

All form components.

Description

Use the *visible* property to conditionally hide a component. You may also disable a component without making it invisible by setting its *enabled* property to *false*. This makes the component appear grayed-out. Depending on the application, it may be more appropriate to completely hide something when it's not needed rather than keeping it visable when the user can do nothing with it.

visibleCount( )

Topic group Related topics

Returns the number of tree items visible in the tree view area.

Syntax

<oRef>.*visibleCount*( )

<oRef>

The tree view to check.

Property of

TreeView

Description

visibleCount( ) returns the number of tree items that are visible in the tree view area. To count all the tree items, whether or not they are visible, use the *count*( ) method.

visualStyle

Topic group Related topics

The style of the tabs in a NoteBook object.

Property of

NoteBook

Description

visualStyle is an enumerated property that can be one of the following values:

Value	Description
0	Right Justify
1	Fixed Width
2	Ragged Right

The Fixed Width style makes all tabs the same width. The Right Justify and Ragged Right Styles are the same when the notebook's *multiple* property is *false*; the tabs are sized to the width of their text label.

When *multiple* is *true* and there are enough tabs to create multiple rows, Right Justify makes the tab edges line up on both the left and right sides, while Ragged Right lines up the tabs on the left edge only.

vScrollBar

Topic group Related topics

Determines when an object has a vertical scroll bar.

Property of

Grid, ListBox

Description

The *vScrollBar* property determines when and if a control displays a vertical scrollbar. It may have any of four settings:

Value	Description
0 (Off)	The object never has a vertical scroll bar.
1 (On)	The object always has a vertical scroll bar.
2 (Auto)	Displays a vertical scroll bar only when needed.
3 (Disabled)	The vertical scroll bar is visible but not usable.

when

Topic group Related topics

Event fired when attempting to give focus to an object; return value determines if object gets focus.

Parameters

<form open expL>

true for when the *when* event handler is called when the form is opened; *false* from then on.

Property of

All form components that can get focus (except NoteBook)

Description

Use *when* to conditionally prevent an object, that has not been disabled, from getting focus. If you set an object's *enabled* property to *false*, the object will appear grayed-out and disabled. This is a visual indication that the object cannot get focus. The object is removed from the tab sequence and clicking the object has no effect.

If an object is enabled, you may define a *when* event handler to determine if an object is available. The event handler must return *true* to allow the object to get focus. If it returns *false*, the object does not get focus. If the object was clicked, focus remains on the object that previously had focus. If Tab or Shift+Tab was pressed, the focus goes to the next control in the tab sequence.

Using *when* gives you the flexibility of displaying a message or taking some other action when the focus is not allowed. But in most cases, it's better to conditionally disable controls so that they are clearly not available. For example, if you have a checkbox to echo output to a file and an entryfield for the file name, you can disable the entryfield when the checkbox is unchecked in the checkbox's *onChange* event handler.

If the *when* event handler returns *true*, or there is no *when* event handler, *onGotFocus* fires after the object receives focus.

The *when* event for all controls is fired when the form first opens. Use the <form open expL> parameter if necessary to distinguish that event from all other normal focus attempts.

Calling an object's *setFocus*( ) method invokes a call to the *when* event handler, and the *when* event's return value determines the success of the *setFocus*( ). The success or failure of the *setFocus*( ) method is not, however, returned to the calling routine.

The firing order for Events when opening a form:

1. The form's *open*( ) method is called
    - The *when* event for each control is fired according to the z-order
    - The form's *onOpen* event is fired
    - The *onOpen* event for each control is fired according to the z-order

The firing order for Form object Events:

Clicking on a form object will result in the following events firing in this order;

2. The *when* event
    - The *onGotFocus* event
    - A mouse event such as, *onLeftDblClick*

Navigating to a form object by using the Tab key will result in the following events firing in this order;

3. The *when* event
    - The *onGotFocus* event

Tip: ON KEY LABEL TAB <command> will perform an action (<command>) when the user presses the TAB key (See ON KEY). However, even though ON KEY LABEL TAB is set to perform <command>, pressing Shift-Tab will still move to another form object (the preceding one in the z-order) and fire it's events in the above order.

width

Topic group Related topics

The width of an object. For Form and SubForm objects, the width of their client areas.

Property of

Form, SubForm and all form contained objects.

Description

Use an object's *width* property in combination with its *height* property to adjust the size of an object.

Form contained objects: The value of the *width* property includes any border, bevel or shadow effect assigned to the object.

Forms and SubForms: The value of the *width* property includes only the client area. It does not include the window border.

When a Form or SubForm is opened and has a vertical scrollbar (see scrollbar property), it's width is automatically reduced by the width of the vertical scroll bar (16 pixels).

When a Form or SubForm's *mdi* property is *true*, the operating system enforces a minimum width of 88 pixels (without a vertical scrollbar) or 104 pixels (with a vertical scrollbar). These enforced minimums will be used should you attempt to set the *width* property to a lower value.

The *width* property is numeric and expressed in the current *metric* unit of the form or subform that contains the object.

The default *metric* unit is "characters", which is the average width of characters in the form's base font (default *scaleFontName* is "MS Sans Serif", default *scaleFontSize* is 8 points). Thus, if the parent forms' *metric* unit is set to characters, and its' *scaleFontName* or *scaleFontSize* properties are changed, the objects on the form are automatically scaled relative to the new font size.

Exception: The *width* property of Line objects is used to set the thickness of the line and is always measured in "pixels".

windowState

Topic group Related topics Example

The maximized/minimized state of the form window.

Property of

Form, SubForm

Description

Use *windowState* to get or set the display state of a form. A window may be maximized, minimized, or in its non-maximized "normal" (restored) state. The appearance of the window also depends on whether it is an MDI window, as noted in the following table.

Setting	Effect on an MDI form	Effect on a non-MDI form
0 (Normal)	If minimized or maximized, the form is restored to its most recent non-maximized size and position within the MDI frame window.	If minimized or maximized, the form is restored to its most recent non-maximized size and position on the screen.
1 (Minimized)	Reduces a normal or maximized form to an iconized title bar at the bottom of the MDI frame window.	Reduces a normal or maximized form to an iconized button on the Windows taskbar.
2 (Maximized)	Enlarges the form to the extent of the MDI frame window.	Enlarges the form to the extent of the screen.

The MDI frame window is the main *dBASE Plus* application window.

If you assign a value to *windowState* that changes its state, the form's *onSize* event fires and the form is also given focus. To give focus to the window without changing its state, call the form's *setFocus*( ) method.

wrap

Topic group Related topics

Determines if an Editor or Text object wraps text automatically.

Property of

ColumnEditor, Editor, Text

Description

Use *wrap* to wrap long lines of text in the editor or in a text object. When *wrap* is *true*, text won't exceed the width of the object and a horizontal scrollbar will not appear. The ColumnEditor, Editor, or Text object will attempt to break each line at a space.

If *wrap* is *false*, long lines extend past the right edge of the Editor. If *scrollBar* is On or Auto, a horizontal scrollbar is used (where needed) to view long lines of text. Even without scrollbars, you can use the cursor to move to the end of long lines.

Report Objects

Report objects

Topic group

Report objects generate formatted output from data in tables. The Report wizard and Report designer allow you to create and modify reports visually. Reports are saved as code in a .REP file that you can modify.

Measurements in reports default to twips (20th of a point). There are exactly 1440 twips per inch.

At the top of the report object class hierarchy is the Report class. A Report object acts as a container for four main groups of objects:

1. Data objects, which give access to data in tables

Query objects
Database objects
Session objects

These objects are created and used the same way they are in forms, except that a report does not have a primary rowset like a form does.

2. Report layout objects, which determine the appearance of the page and where data is output, or streamed

PageTemplate objects
StreamFrame objects

A Report object contains one or more PageTemplates, and each PageTemplate usually contains one or more StreamFrames.

3. Data stream objects, which read and organize the data from a query's rowset and stream it out to a report's StreamFrame objects

StreamSource objects
Band objects
Group objects

Each StreamSource object contains a Band object that is assigned to its *detailBand* property. The contents of the *detailBand* are rendered for each row in the rowset. A StreamSource may also have one or more Group objects, which group data and have their own header and footer Band objects.

4. Visual components—objects that display the report's data

Text objects
Image objects
Line objects
Rectangle objects
Shape objects
CheckBox objects
RadioButton objects

These objects are created as properties of a PageTemplate object if they are fixed elements on the page, such as a report's date and page number; otherwise they are properties of a Band object and are used to display data.

The primary method of displaying information in a report is through Text objects. For text that varies, such as the data from the rowset, the *text* property of the Text object is set to an expression codeblock, which is evaluated every time the object is rendered. By using an expression in the codeblock that accesses the fields in the rowset, the Text object displays data from tables.

You may use the other visual components in a report to display static images or images from a table, draw lines, or display table data with check boxes or RadioButtons.

Note

Visual component objects are used in forms as well as reports, and most of the properties, methods, and events associated with the objects are described in the Form objects series of topics. Some Text object properties used only in reports are described in this series.

A simple report example

Topic group Related topics

To get a sense of how everything fits together, imagine a report of students grouped by grade, with the total number of students in each grade.

The report has a query that accesses the table of students, named students1; a StreamSource object, by default named streamSource1, to stream the data from the query; and a PageTemplate object, by default named pageTemplate1, that describes the physical attributes of the page, such as its dimensions, background color, and margins.

pageTemplate1 contains one StreamFrame object, by default named streamFrame1, where the data stream will be rendered. It occupies most of the space inside pageTemplate1's margins. The rest of the space is used by Text components that display the report title, date, and page.

streamFrame1 has a *streamSource* property that identifies its StreamSource object. It is assigned streamSource1.

streamSource1 has a *rowset* property that identifies the StreamSource object's rowset. It is assigned students1.*rowset*.

students1.*rowset* and streamFrame1 are now linked. To fill streamFrame1 with data, the report engine will traverse students1.*rowset*, from the first row to the last row. But at this point, no data will be displayed, because there are no visual components in any Band objects.

Text components are assigned to streamSource1.*detailBand*. The *text* properties of these components are expression codeblocks that refer to the *value* properties of the fields of the *rowset* of the StreamSource object. For example, the *text* of the Text component that displays the student's last name is

```
{||this.form.students1.rowset.field[ "Last name" ].value}
```

When a visual component is placed in a report, its *form* property refers to the report.

To group the data, a Group object, named group1 by default, is assigned to streamSource1. Its *groupBy* property contains the name of the group field, "Grade". The report engine will watch the value of this field in the rowset, that is:

```
students1.rowset.field[ "Grade" ].value
```

and whenever the value of the field changes, a new group begins. Therefore, it's important that the data is sorted by grade. If the report's *autoSort* property is *true*, all of the report's queries will automatically be sorted to match the groups in the StreamSource objects.

group1 has two Band objects of its own: a header band and a footer band, assigned to the *headerBand* and *footerBand* properties respectively. The *headerBand* is currently empty, and the *footerBand* displays the count of the students in that grade.

The Group object's *agCount*( ) method counts the number of rows in the group. To display that number, the *text* of the Text component in the *footerBand* is set to the following expression codeblock:

```
{||"Count: " +
this.parent.parent.agCount({||this.parent.rowset.fields["ID"].value})}
```

The expression codeblock concatenates the text label with the return value of the Group object's *agCount*( ) method. To get to that method from a component in the *footerBand*,

this is the component.

The component's *parent* is the *footerBand*.

The *footerBand*'s *parent* is the Group.

The *agCount*( ) method expects a code reference as a parameter that it can evaluate. If the return value is not *null*, the count is incremented. The code reference here is another expression codeblock that uses dot operators:

this is the Group object group1.

group1's *parent* is streamSource1.

streamSource1's *rowset* is students1.*rowset*, the rowset that the report engine is traversing to fill streamFrame1.

That's all the objects that go into a report of students, grouped by grade, with the number of students in each grade. There are two final details that are needed to make the report work.

Because a report can have multiple PageTemplate objects, a Report object has a *firstPageTemplate* property that refers to the PageTemplate object to use for the first page. It is assigned pageTemplate1.

Each PageTemplate object has a *nextPageTemplate* property that refers to the PageTemplate object to use when the current page is done. For pageTemplate1, it is assigned a reference to itself. This means that the same page layout is used for every page in the report.

Everything described in this sample report can be handled automatically by the Report Wizard. To run the report, call the Report object's *render*( ) method.

How a report is rendered

Topic group Related topics

When a Report object's *render*( ) method is called, the first thing the report does is check its *firstPageTemplate* property to find the first page to render. It renders the page by rendering all the components and StreamFrame objects assigned to it, in the order they were originally created (the same order as they appear in the class definition in the .REP file).

To render a StreamFrame object, *dBASE Plus* looks to its *streamSource* property. The Band objects in that StreamSource object—the *detailBand* and the *headerBand* and *footerBand* of any groups—are rendered in the StreamFrame object to fill it with data.

Before each component in the band is rendered, its *canRender* event fires. The *canRender* event can be used to supplement the *suppressIfBlank* and *suppressIfDuplicate* properties of the Text component by returning *false*, but it is more often used to alter the properties of a component just before it is rendered. For example, you can set a component's *colorNormal* to red if it's going to display a negative number. When used this way, the *canRender* event handler does what it wants and returns *true*, so that component is rendered. After the component is rendered, its *onRender* event fires. You can use the *onRender* event to reset the component to its original state.

Until the data from the StreamSource object is exhausted—that is unless the StreamSource object's *rowset* reaches the end-of-set—*dBASE Plus* knows that it needs to fill another StreamFrame. If there is another StreamFrame object in the same PageTemplate that used the same *streamSource*, the report engine will continue to stream bands from that StreamSource into that StreamFrame.

For example, if a PageTemplate has three tall StreamFrame objects side-by-side that have the same *streamSource* property, data would be printed in three columns on each page. To create a page of labels, create one StreamFrame for each label, all with the same *streamSource* property. Then set the *beginNewFrame* property of the *streamSource*'s *detailBand* to *true*, so that each row of data is rendered in a new StreamFrame.

If there are no more StreamFrame objects that can be filled on the current page, another page is scheduled. The current PageTemplate object's *nextPageTemplate* property refers to the PageTemplate to use.

Once the current page has finished rendering, the Report object's *onPage* event fires. If there is another page scheduled, it is rendered. Its StreamFrame objects are filled with data and the process repeats itself until all the StreamSource objects are exhausted. The *onPage* event fires one last time and the report is done.

class Band

Topic group Related topics

Contains the objects to output for a single row in a stream, or the header or footer of a group.

Syntax

These objects are automatically created by the StreamSource and Group objects.

Properties

The following table lists the properties and events of the Band class. (No methods are associated with this class.)

Property	Default	Description
baseClassName	BAND	Identifies the object as an instance of the Band class
beginNewFrame	false	Whether rendering always starts in a new StreamFrame
className	(BAND)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
context	Normal	The context in which the band is being rendered: (0=Normal) or for (1=For Drilldown summary)

expandable	true	Whether the band will increase in size automatically to accommodate the objects within it
firstOnFrame		Whether the band is being rendered for the first time in a StreamFrame.
height	0	The height of the band in the Report object's current metric units
name		The name of the Band object
parent		The StreamSource or Group object that contains the Band
renderOffset		The offset of the bottom of the band from the top of the current stream frame.
streamFrame		The StreamFrame object in which the band is currently rendering.
visible		Whether the band is visible

Event	Parameters	Description
onRender		After the contents of the band have rendered
preRender		Before the contents of the band are rendered

Description

A Band object acts as a container for visual components. They are created automatically for StreamSource and Group objects and cannot be created manually. There are three kinds of Band objects: detail bands, header bands, and footer bands.

A detail band is assigned to a StreamSource's *detailBand* property. The contents of the band are output once for each row in the StreamSource's rowset. Header and footer bands are assigned to a Group object's *headerBand* and *footerBand* properties respectively. They are rendered at the beginning and end of each group.

For a detail band, setting its *beginNewFrame* property to *true* causes each row from the StreamSource's rowset to be rendered in a new StreamFrame, which is the desired behavior when creating labels.

For a summary-only report, leave the detail band empty and set its *height* to zero.

When a band's *expandable* property is *true* and it contains components, the band will expand to show those components, even if its *height* is set to zero.

class Group

Topic group Related topics

Describes a group in a report.

Syntax

[<oRef> =] new Group(<streamSource>)

<oRef>

A variable or property—typically of <streamSource>—in which you want to store a reference to the newly created Group object.

<streamSource>

The StreamSource object to which the Group object binds itself.

Properties

The following tables list the properties and methods of the Group class. (No events are associated with this class.)

Property	Default	Description
baseClassName	GROUP	Identifies the object as an instance of the Group class.
className	(GROUP)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
drillDown	None	How the group's bands are displayed in drilldown format.
footerBand		Specifies a Band that renders after a group of detail bands.
groupBy		A character string containing the field name by which groups are formed. If blank, the group is for the entire report.
headerBand		Specifies a Band that renders before a group of detail bands.
headerEveryFrame	false	Specifies whether to repeat the *headerBand* when a Group spans more than one StreamFrame.
name		The name of the Group object.
parent		The Report or StreamSource object that contains the Group.

Method	Parameters	Description
agAverage( )	<codeblock>	Aggregate method that returns the mean average for a group
agCount( )	<codeblock>	Aggregate method that returns the number of items in a group
agMax( )	<codeblock>	Aggregate method that returns the highest value within a group
agMin( )	<codeblock>	Aggregate method that returns the lowest value in a group
agStdDeviation( )	<codeblock>	Aggregate method that returns the standard deviation of the values in a group
agSum( )	<codeblock>	Aggregate method that returns the total of a group
agVariance( )	<codeblock>	Aggregate method that returns the variance of the values in a group
release( )		Explicitly releases the Group object from memory

Description

Use Group objects to group data and calculate aggregate values for the group. Groups may be nested, and are handled in the order that they are created (the same order that they appear in the class definition in a .REP file).

The *groupBy* property contains the name of the field that defines the group, and may include an optional ascending or descending modifier. Whenever the value of that field changes, a new group starts. Therefore, the data must be sorted on the grouping field(s).

A Group object's *headerBand* is rendered before each group and its *footerBand* is rendered afterward. If the *headerEveryFrame* property is *true*, the group's *headerBand* is rendered at the beginning of every StreamFrame.

If the Report object's *autoSort* property is true, data in a report is automatically sorted to match groups.

The Report object has its own Group object that is referred to by its *reportGroup* property. Its *groupBy* property is an empty string, and the group is used for report-wide aggregates.

You may organize the report in drilldown format: the header and footer bands showing summary information are displayed first, followed by the detail rows. This allows you to see summary information at the top, and then "drill down" to the supporting data.

class PageTemplate

Topic group Related topics

Describes the layout of a page of a report.

Syntax

[<oRef> =] new PageTemplate(<report>)

<oRef>

A variable or property—typically of <report>—in which you want to store a reference to the newly created PageTemplate object.

<report>

The Report object to which the PageTemplate object binds itself.

Properties

The following tables list the properties and methods of the PageTemplate class. (No events are associated with this class.)

Property	Default	Description
background		Background image on the page
baseClassName	PAGETEMPLATE	Identifies the object as an instance of the PageTemplate class
className	(PAGETEMPLATE)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
colorNormal	white	Background color for the page
gridLineWidth	0	Width of lines around elements in the report (0=no grid lines)
height		Height of the page in current metric units
marginBottom	.75 inch = 1080 twips	The space between the bottom of the page and the usable area of the PageTemplate
marginLeft	.75 inch = 1080 twips	The space between the left side of the page and the usable area of the PageTemplate
marginRight	.75 inch = 1080 twips	The space between the right side of the page and the usable area of the PageTemplate
marginTop	.75 inch = 1080 twips	The space between the top of the page and the usable area of the PageTemplate
name		The name of the PageTemplate object
nextPageTemplate		The PageTemplate object that is used for the following page
parent		The Report object that contains the PageTemplate
width		Width of the page in current metric units

Method	Parameters	Description
release( )		Explicitly releases the PageTemplate object from memory

Description

A PageTemplate object describes the layout of a page, including its background color or image. It acts as a container for StreamFrame objects and visual components, which represent fixed output, such as a report date and page number.

The location of these objects is relative to (and restricted by) the four margin- properties that dictate the usable area of the page. Changing the *marginLeft* or *marginTop* will move everything

that's inside the PageTemplate. A PageTemplate's dimensions - it's height and width - correspond to your printer's current Paper Size settings.

Although you may create multiple PageTemplate objects in a report, for example a different first page or alternating odd and even pages, the Report Designer currently does not support multiple PageTemplate objects visually.

In the Report Designer, the dotted area represents the useable portion of the PageTemplate, with the surrounding white area indicating the margins. The Report Designer shows only that area which corresponds to a PageTemplate.

class Report

Topic group Related topics

A container and controller of report elements.

Syntax

[<oRef> =] new Report( )

<oRef>

A variable or property in which you want to store a reference to the newly created Report object.

Properties

The following tables list the properties, events, and methods of the Report class.

Property	Default	Description
autoSort	true	Whether to automatically sort data to match specified groups
baseClassName	REPORT	Identifies the object as an instance of the Report class
className	(REPORT)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
endPage	–1	Last page number to render (–1 for no limit)
elements		An array containing object references to the visual components on the reports
firstPageTemplate		Reference to the first PageTemplate object, which describes the first page
inDesign		Whether the report was instantiated by the Report designer
mdi		Whether the report window is an MDI window
metric	Twips	Units of measurement (0=Chars, 1=Twips, 2=Points, 3=Inches, 4=Centimeters, 5=Millimeters, 6=Pixels)
output	Default	Target media (0=Window, 1=Printer, 2=Printer file, 3=Default, 4=HTML file, 5=CGI Response)
outputFilename		Name of file if output goes to printer, HTML file, or CGI
printer		An object describing various printer output options
reportGroup		Reference to a Group object for the report as a whole, for master counts and totals
reportPage		Current page number being rendered
reportViewer	null	Reference to the ReportViewer object that instantiated the report, if any.
scaleFontBold	false	When the *metric* is Chars, determines whether the Char units of the ScaleFont assume that the font is bold

scaleFontName	Arial	When the *metric* is Chars, the typeface of the font used as the basis of measurement
scaleFontSize	10	When the *metric* is Chars, the point size of font used as the basis of measurement
startPage	1	First page number to output
title		Title of the report; appears in the title bar of the preview window
usePrintingDialog	true	Whether or not report's Printing Dialog willt be displayed while rendering report.

Event	Parameters	Description
onDesignOpen		After the report is first loaded into the Report Designer
onPage		After a page is rendered

Method	Parameters	Description
close( )		Closes the report window
isLastPage( )		Determines whether there are any more pages to render
release( )		Explicitly releases the Report object from memory
render( )		Generates the report

Description

A Report object acts as the controlling container for all the objects that make up the report, including data access, page layout, and data stream objects.

The *reportGroup* property refers to a report-level Group object that can be used for report-wide summaries. This Group object is created automatically.

To generate the report, call its *render*( ) method. The report's *output* property determines where the report is rendered: to the screen, a printer, or a file. The report's *printer* object contains properties that control output to a printer (or printer file). Call the *printer* object's *choosePrinter*( ) method before calling *render*( ) to allow the user to choose a printer.

You can control the pages that are output by setting the *startPage* and *endPage* properties.

class StreamFrame

Topic group Related topics

Describes an area on a page into which output is streamed.

Syntax

[<oRef> =] new StreamFrame(<pageTemplate>)

<oRef>

A variable or property—typically of <pageTemplate>—in which you want to store a reference to the newly created StreamFrame object.

<pageTemplate>

The PageTemplate object to which the StreamFrame object binds itself.

Properties

The following table lists the properties and events of the StreamFrame class. (No methods are associated with this class.)

Property	Default	Description
baseClassName	STREAMFRAME	Identifies the object as an instance of the StreamFrame class
borderStyle	Default	The border around the StreamFrame object (0=Default, 1=Raised, 2=Lowered, 3=None, 4=Single, 5=Double, 6-Drop Shadow, 7=Client, 8=Modal, 9=Etched In, 10=Etched Out)
className	(STREAMFRAME)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
form		Reference to the report that contains the StreamFrame object
height	0	Height of the StreamFrame object in its PageTemplate's Report's current metric units
left	0	The location of the left edge of the StreamFrame object in its PageTemplate's Report's current metric units, relative to the PageTemplate's *marginLeft*
marginHorizontal	0	Horizontal margin inside the StreamFrame
marginVertical	0	Vertical margin inside the StreamFrame
name		The name of the StreamFrame object
parent		The PageTemplate object that contains the StreamFrame
streamSource		Reference to a StreamSource object that contains objects to be rendered in the StreamFrame
top	0	The location of the top edge of the StreamFrame object in its PageTemplate's Report's current metric units, relative to the PageTemplate's *marginTop*
width		Width of the StreamFrame object in its PageTemplate's Report's current metric units

Event	Parameters	Description
canRender		Before the StreamFrame is rendered; return value determines whether StreamFrame is rendered
onRender		After the contents of the StreamFrame have rendered

Description

A StreamFrame object describes a rectangular region inside the margins of a PageTemplate into which data from a StreamSource object is rendered.

Although you may create multiple StreamFrame objects in a PageTemplate, the Report Designer currently does not support multiple StreamFrame objects visually.

class StreamSource

Topic group Related topics

Describes a data source for streaming.

Syntax

[<oRef> =] new StreamSource(<report>)

<oRef>

A variable or property—typically of <report>—in which you want to store a reference to the newly created StreamSource object.

<report>

The Report object to which the StreamSource object binds itself.

Properties

The following tables list the properties and methods of the StreamSource class. (No events are associated with this class.)

Property	Default	Description
baseClassName	STREAMSOURCE	Identifies the object as an instance of the StreamSource class
className	(STREAMSOURCE)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
detailBand		A Band object that corresponds to the *rowset*
maxRows		The maximum number of rows the StreamSource will provide per StreamFrame per page
name		The name of the StreamSource object
parent		The Report object that contains the StreamSource
rowset		The Rowset object that drives the StreamSource

Method	Parameters	Description
beginNewFrame( )		Forces the next band to display in a new StreamFrame.
release( )		Explicitly releases the StreamSource object from memory

Description

A StreamSource object acts as the common ground between a rowset that contains data you want to display and a band that contains components to display that data.

Every StreamFrame is assigned a StreamSource. The same StreamSource object may be assigned to multiple StreamFrame objects. The data from a StreamSource is rendered in all the StreamFrame objects that are linked to it. You may limit the number of rows that are rendered per StreamFrame, and therefore per page, by setting the StreamSource object's *maxRows* property.

A StreamSource object may contain Group objects that group data to perform aggregate functions.

agAverage( )

Topic group Related topics Example

Aggregate method that returns the mean average for a group.

Syntax

<oRef>.*agAverage*(<codeblock>)

<oRef>

The Group object that defines the group you want to summarize.

<codeblock>

A codeblock or pointer to a function that returns the value to average.

Property of

Group

Description

Use *agAverage*( ) to calculate the mean average of the value returned by <codeblock> in the group. <codeblock> is usually an expression codeblock that returns the *value* property of a field in the Group object's *parent* StreamSource object's *rowset*.

If <codeblock> returns a *null* value, it is not considered in the average.

You may call *agAverage*( ) at any time. If necessary, the report engine will look ahead to calculate the result.

agCount( )

Topic group Related topics Example

Aggregate method that returns the number of items in a group.

Syntax

<oRef>.*agCount*(<codeblock>)

<oRef>

The Group object that defines the group you want to summarize.

<codeblock>

A codeblock or pointer to a function that returns the value you want to count.

Property of

Group

Description

Use *agCount*( ) to count the number of items in the group. <codeblock> is usually an expression codeblock that returns the *value* property of a field in the *rowset* of the Group object's *parent* StreamSource object.

If <codeblock> returns a *null* value, that item is not counted, so that empty rows will be skipped. To count a row even if it is empty, have the <codeblock> return a constant non-*null* value, for example,

```
{||1}
```

You may call *agCount*( ) at any time. If necessary, the report engine will look ahead to calculate the result.

agMax( )

Topic group Related topics Example

Aggregate method that returns the highest value within a group.

Syntax

<oRef>.*agMax*(<codeblock>)

<oRef>

The Group object that defines the group you want to summarize.

<codeblock>

A codeblock or pointer to a function that returns the value you want to track.

Property of

Group

Description

Use *agMax*( ) to return the highest value returned by <codeblock> in the group. <codeblock> is usually an expression codeblock that returns the *value* property of a field in the *rowset* of the Group object's *parent* StreamSource object.

If <codeblock> returns a *null* value, it is ignored.

You may call *agMax*( ) at any time. If necessary, the report engine will look ahead to determine the result.

agMin( )

Topic group Related topics Example

Aggregate method that returns the lowest value within a group.

Syntax

<oRef>.*agMin*(<codeblock>)

<oRef>

The Group object that defines the group you want to summarize.

<codeblock>

A codeblock or pointer to a function that returns the value you want to track.

Property of

Group

Description

Use *agMin*( ) to return the lowest value returned by <codeblock> in the group. <codeblock> is usually an expression codeblock that returns the *value* property of a field in the *rowset* of the Group object's *parent* StreamSource object.

If <codeblock> returns a *null* value, it is ignored.

You may call *agMin*( ) at any time. If necessary, the report engine will look ahead to determine the result.

agStdDeviation( )

Topic group Related topics Example

Aggregate method that returns the standard deviation of the values in a group.

Syntax

<oRef>.*agStdDeviation*(<codeblock>)

<oRef>

The Group object that defines the group you want to summarize.

<codeblock>

A codeblock or pointer to a function that returns the value you want to sample.

Property of

Group

Description

Use *agStdDeviation*( ) to calculate the standard deviation of the value returned by <codeblock> in the group. <codeblock> is usually an expression codeblock that returns the *value* property of a field in the *rowset* of the Group object's *parent* StreamSource object.

If <codeblock> returns a *null* value, it is not considered in the sample.

You may call *agStdDeviation*( ) at any time. If necessary, the report engine will look ahead to calculate the result.

agSum( )

Topic group Related topics Example

Aggregate method that returns the total of a group.

Syntax

<oRef>.*agSum*(<codeblock>)

<oRef>

The Group object that defines the group you want to summarize.

<codeblock>

A codeblock or pointer to a function that returns the value you want to total.

Property of

Group

Description

Use *agSum*( ) to calculate the total of the value returned by <codeblock> in the group. <codeblock> is usually an expression codeblock that returns the *value* property of a field in the *rowset* of the Group object's *parent* StreamSource object.

If <codeblock> returns a *null* value, it is ignored.

You may call *agSum*( ) at any time. If necessary, the report engine will look ahead to calculate the result.

agVariance( )

Topic group Related topics Example

Aggregate method that returns the variance of the values in a group.

Syntax

<oRef>.*agVariance*(<codeblock>)

<oRef>

The Group object that defines the group you want to summarize.

<codeblock>

A codeblock or pointer to a function that returns the value you want to sample.

Property of

Group

Description

Use *agVariance*( ) to calculate the variance of the value returned by <codeblock> in the group. <codeblock> is usually an expression codeblock that returns the *value* property of a field in the *rowset* of the Group object's *parent* StreamSource object.

If <codeblock> returns a *null* value, it is not considered in the sample.

You may call *agVariance*( ) at any time. If necessary, the report engine will look ahead to calculate the result.

autoSort

Topic group Related topics

Whether to automatically sort data to match specified groups.

Property of

Report

Description

For groups to work properly, data must be sorted to match the groups.

If a Report object's *autoSort* property is *true*, then the *sql* property of any query that is accessed by a StreamSource object that has groups will be modified automatically to include an ORDER BY clause that sorts the rowset in the correct order.

For example, if you have two Group objects, the first grouping by the field State and the second by Zip, then even if the query's *sql* property is set as:

```
select * from SALES
```

the rowset will actually be generated internally with the SQL statement:

```
select * from SALES order by STATE, ZIP
```

If *autoSort* is *false*, the rowset is not altered by the report engine. It assumes that the query is correct and contains the necessary fields in the right order. Therefore, if you use the *indexName* property to set the rowset order, you should set *autoSort* to *false*; otherwise it defeats the purpose of using *indexName*.

Note: The default for the *autoSort* property was changed to *false* in dBASE Plus version 2.50. Versions prior to this had *autoSort* defaulting to *true*.

beginNewFrame

Topic group Related topics

Specifies whether rendering always starts in a new StreamFrame.

Property of

Band

Description

Set the *beginNewFrame* property of the StreamSource object's *detailBand* to *true* if you want each row to be rendered in its own StreamFrame. If you have one StreamFrame in each

PageTemplate, you will get one row per page. If you have multiple StreamFrames in each PageTemplate, each one will have at most one row of data.

You would create a page of labels by creating a StreamFrame for each label, set all the StreamFrame objects' *streamSource* property to the same StreamSource, and set its *detailBand*'s *beginNewFrame* property to *true*.

Set the *beginNewFrame* property of a group's *headerBand* to *true* if you want each group to start in a new StreamFrame. If you have one StreamFrame per page, that makes each group start on a new page.

If the *beginNewFrame* property of a group's *footerBand* is *true*, then whenever it is rendered, it will start in a new StreamFrame. For example, you could print a summary page for a report by creating a large *footerBand* for the Report object's *reportGroup* and set its *beginNewFrame* property to *true*.

beginNewFrame( )

Topic group Related topics Example

Forces the next band to display in a new StreamFrame.

Property of

StreamSource

Description

Use *beginNewFrame*( ) to conditionally force the next band—whether it is a *detailBand*, *headerBand*, or *footerBand*—to display in a new StreamFrame. In effect, it's as if that band's *beginNewFrame* property is temporarily set to true.

canRender

Topic group Related topics Example

Just before the component is rendered; return value determines whether the component is displayed.

Parameters

none

Property of

All visual components on a report, StreamFrame

Description

canRender fires for visual components only when they are in a report. It is fired every time the object is rendered. For a component in a detail band, that means for every row in the rowset.

While you can use *canRender* to evaluate some condition and return *false* to prevent the component from being displayed, the more common use of *canRender* is to alter a component's properties conditionally and always return *true*. You can create a calculated field in a report by altering an HTML component's *text* property in its *canRender* event handler.

You can use the *onRender* event to reset the component to its default state afterward, or always choose the desired state in the *canRender* event.

For a StreamFrame object, the *canRender* event fires before it attempts to retrieve data from its *streamSource*. Note that it is the rendering of StreamFrame objects that cause additional pages to be scheduled. If a report has only one stream frame, and its *canRender* returns *false* so that it is not rendered, no more pages will be printed; the report will terminate. You can call *streamSource.beginNewFrame*( ) to skip the current stream frame, but in that case, its *canRender* event handler must return *true*.

choosePrinter( )

Topic group

Opens a print setup or print dialog to allow a user to select the printer and set other print or report properties.

Syntax

<oRef>.*choosePrinter*([<title expC>][, <expL>])

<oRef>

A reference to the printer object whose *choosePrinter*( ) method you wish to call.

<title expC>

Optional custom title for the print or printer setup dialog box.

<expL>

If true, *choosePrinter*( ) will display the "Print Setup" dialog.

If false, *choosePrinter*( ) will display the standard Windows "Print" dialog.

Property of:

printer class

Description

When calling the *choosePrinter*( ) method of the _app.printer object, the "Print Setup" dialog will display, by default.

When calling the *choosePrinter*( ) method of a printer object that is attached to a report object, the standard Print dialog will display, by default. If the user sets a start page and end page to print, their settings will be copied into the attached report's *startPage* and *endPage* properties.

If a report object's *startPage* and *endPage* values are already set when the associated *choosePrinter*( ) method is called, these values will be defaulted into the Print dialog.

For either dialog, user selections will be copied into the corresponding printer object properties

context

Topic group Related topics

Reports the context in which the band is being rendered.

Property of

Band

Description

Check the band's *context* property to determine the context in which it's being rendered. *context* is a read-only enumerated property that can have the following values:

Value	Context
0	Normal
1	For drilldown summary

For detail bands, the value will always be Normal, since they are never rendered in the drilldown summary. For footer and header bands, the value will change accordingly.

The *context* property is checked primarily during the *canRender* event of components in a header or footer band when the header or footer is shown in both the summary and details in a drilldown report; you can change the content of the component accordingly.

detailBand

Topic group Related topics

The Band object in a StreamSource, which displays data from the rowset.

Property of

StreamSource

Description

A StreamSource object automatically has a Band object assigned to its *detailBand* property. This is the band that is rendered to display data in the rowset.

Visual components for displaying detail rows in the report should be created as a property of the StreamSource object's *detailBand*.

drillDown

Topic group Related topics

How the group's bands are displayed in drilldown format.

Property of

Group

Description

By choosing a drilldown format, the header and footer bands in the specified group are displayed first, followed by the detail rows. The *drillDown* property controls whether the header and footer bands are repeated with the detail rows. *drillDown* is an enumerated property that can have one of the following values:

Value	Description
0	None—do not format as drilldown (default)
1	Drilldown. Do not repeat headers or footers
2	Drilldown, repeat headers
3	Drilldown, repeat footers
4	Drilldown, repeat headers and footers

endPage

Topic group Related topics

The last page number to render.

Property of

Report

Description

When rendering to a window (the default), only one page, the *startPage*, is rendered at a time. When rendering to a printer or file, pages are rendered from the *startPage* to the *endPage*.

By default, *endPage* is –1, which means that all the pages in the report are rendered. Set *endPage* to a number greater than zero to set the last page to render. When and if the report engine gets to that page, it stops after it has finished rendering it.

expandable

Topic group Related topics

Specifies whether an object will increase in size automatically to accommodate the objects within it.

Property of

Band

Description

If a Band object's *expandable* property is *true* (the default), it will increase in size to display all the components inside it, even if its *height* is set to zero.

Set *expandable* to *false* if you want to make the size and number of rows displayed on each page constant, no matter what is displayed.

firstOnFrame

Topic group Related topics

Whether the band is being rendered for the first time in the StreamFrame.

Property of

Band

Description

Check the *firstOnFrame* property in a component's *canRender* event if you want to render it (or don't want to render it) the first time the component's band is being rendered in a StreamFrame only.

Example

If you place column headings inside a StreamFrame, you can create Text components in the detail band that have the following *canRender* event:

```
{||this.parent.firstOnFrame}
```

firstPageTemplate

Topic group Related topics

The PageTemplate object that is used for the report's first page.

Property of

Report

Description

Because a report may have multiple PageTemplate objects, the *firstPageTemplate* property is used to identify the PageTemplate that the report should render as its first page.

Once the first PageTemplate has been chosen, each PageTemplate object has a *nextPageTemplate* property that identifies the page to render next.

fixed

Topic group Related topics

Specifies whether an object's position in a band is fixed or if it can be "pushed down" or "pulled up" by the rendering or suppression of other objects.

Property of

Visual components in a band.

Description

Consider two components in a band named object1 and object2. Suppose that

The bottom of object1 is at or above the *top* of object2.
object1's *variableHeight* property is *true*.
object1 grows in height to accommodate the data in it.
The bottom of object1 is now below the *top* of object2.

Then if object2's *fixed* property is *false* (the default), object2 will be pushed down by object1 so that object2's *top* will be at the bottom of object1. This in turn might push down other objects in the band.

object2 can also be pulled up if the bottom of object1 is at or above the top of object2 and object1 is suppressed by its *suppressIfBlank* or *suppressIfDuplicate* properties.

The horizontal position of the objects in question doesn't matter, only the vertical position of their top and bottom ends.

If an object's *fixed* property is *true*, it is not moved by the rendering or suppression of other objects.

After the band has been rendered, all the objects return to their original positions and sizes.

footerBand

Topic group Related topics Example

The Band object that renders after each group.

Property of

Group

Description

A Group object automatically has a Band object assigned to its *footerBand* property. This band is rendered after each group is completed; that is, just before the next group starts or at the end of the rowset. It usually contains components that display summary information.

For components in the *footerBand*, the Group object's aggregate functions will return summary values for the group that has just completed.

gridLineWidth

Topic group

The pageTemplate's *gridLineWidth* property sets the border attribute of the table used for HTML reports. It does not affect reports sent to the printer.

Property of

PageTemplate

Description

The value entered for the *gridLineWidth* property will be streamed out to the definition of the table that defines the layout of the report.

groupBy

Topic group Related topics Example

The name of the field upon which groups are formed.

Property of

Group

Description

dBASE Plus groups rows by monitoring the value of a field in the rowset. The *groupBy* property contains the name of a field as a character string with an optional ascending or descending sort designation (not case-sensitive, ascending is the default). You may abbreviate ascending as "ASC" and descending as "DESC". For example, if you're grouping by the Age field, you could have one of the following strings as the *groupBy* property:

```
Age
Age asc
Age desc
Age ascending
Age descending
```

The ascending and descending options have an effect only if the report's *autoSort* property is set to *true*. In that case, *dBASE Plus* will make sure that the data in the rowset is sorted by the correct field in the correct direction.

No matter what the value of *autoSort* is, the named field must exist in the rowset. *dBASE Plus* looks for that field in the rowset's *fields* array, just as you would. For example,

```
rowset.fields[ "Age" ].value
```

Because *dBASE Plus* uses the rowset's *fields* array, you can group on calculated fields. There are two ways to do this. You can create a calculated field in an SQL SELECT statement, in which case set *autoSort* to *true*; or you can create the calculated field by adding a Field object to the rowset's *fields* array, in which case you must set *autoSort* to *false*, because the named field

doesn't exist in the table, so you can't sort on it. You would still have to make sure that the rows are sorted in the correct order so that all the rows in the same group are contiguous in the rowset.

headerBand

Topic group Related topics Example

The Band object that renders before each group.

Property of

Group

Description

A Group object automatically has a Band object assigned to its *headerBand* property. This band is rendered before the start of a new group, including the first group in the report; and if the Group object's *headerEveryFrame* property is true, at the beginning of every StreamFrame. It usually contains components that identify the current group or display summary information.

headerEveryFrame

Topic group Related topics

Specifies whether to repeat a group's *headerBand* when a group spans more than one StreamFrame.

Property of

Group

Description

A group's *headerBand* is rendered at the beginning of the group. By default, *headerEveryFrame* is *false*; that means that if the contents of the group go into another frame, the *headerBand* is not repeated.

Set *headerEveryFrame* to *true* if you want a group's *headerBand* to be rendered at the top of every StreamFrame. For example, if you have one StreamFrame per page, setting *headerEveryFrame* to *true* will print the *headerBand* at the top of each page, underneath the fixed components (for example column headings) on the page.

If you have nested groups with *headerEveryFrame* set to *true* for each *headerBand*, the header bands will appear in group order at the top of every StreamFrame.

The headerBand's *beginNewFrame* property determines whether the header band for a new group should start in a new StreamFrame. In contrast, *headerEveryFrame* determines whether the header band should be repeated in subsequent StreamFrame objects.

isLastPage( )

Topic group Related topics Example

Returns *true* or *false* to let you know if additional pages are due to be rendered.

Syntax

<oRef>.*isLastPage*( )

<oRef>

An object reference to the report you want to check.

Property of

Report

Description

isLastPage( ) returns *true* if the current page is the last page of the report, and *false* if additional pages are to be rendered.

Its main purpose is to allow you to make informed decisions about whether or not to display a button to display additional pages. You may also use *isLastPage*( ) to display something on the last page of a report.

dBASE Plus does not determine in advance how many pages a report will take. It renders the report one page at a time by filling all the StreamFrame objects on that page with data drawn from the StreamFrame objects' *streamSource*. If there is more data to render and all the StreamFrame objects in the page are full, another page is scheduled.

If the page being rendered is before the report's *startPage*, the rendering is not output. When rendering to a window (the default), rendering stops once a page is output; the window only displays one page at a time. Rendering to a printer or file renders multiple pages. After the page has finished rendering, if another page is scheduled, it is rendered. The process repeats until all the pages are rendered, or the report's *endPage* page is rendered. In that case, the rendering process stops, even though another page may be scheduled.

isLastPage( ) ignores the *endPage* setting and determines if another page is scheduled to be rendered. It can be reliably called only after the last StreamFrame on a PageTemplate has been rendered, since it is the rendering of StreamFrame objects that determines the scheduling of new pages.

isLastPage( ) is usually called from the *canRender* event handler for a component attached to the PageTemplate—not in a band—that is defined after all the StreamFrame objects. The order in which objects are created and assigned in the report class constructor directly determines their order of definition and rendering.

leading

Topic group Related topics

The distance between consecutive lines inside a component.

Property of

Text

Description

leading controls the line spacing within an Text component. By default, it's zero, which uses the default line space for the font.

You can set *leading* to a non-zero value to set the baseline-to-baseline distance of the text in the component.

marginBottom

Topic group Related topics

The space between the bottom of the page and the usable area of the PageTemplate.

Property of

PageTemplate

Description

Use *marginBottom* in conjunction with the PageTemplate's other margin- properties to define the usable area of the page. The position of StreamFrame objects and components bound to the PageTemplate is relative to the top left corner of the usable area and cannot extend beyond the bottom right corner.

marginBottom indicates the distance, in the report's current *metric* units, between the bottom of the page and the bottom of the usable area.

When using multiple PageTemplate objects, you can position items differently just by changing the margins. For example, you may use different left and right pages that have increased margins on the inside edge (the gutter space) for binding.

marginHorizontal

Topic group Related topics

The horizontal margin between the edge of the object and its contents. For use in Reports only.

Property of

Editor, Text, StreamFrame

Description

An object's horizontal margin is the same on both the left and right sides. In a Text component, the text is indented inside its rectangular frame. The *left* position of all bands inside a StreamFrame object are relative to the horizontal margin on the left and restricted by the horizontal margin on the right.

marginHorizontal is measured in the form or report's current *metric* units.

marginLeft

Topic group Related topics

The space between the left edge of the page and the usable area of the PageTemplate.

Property of

PageTemplate

Description

Use *marginLeft* in conjunction with the PageTemplate's other margin- properties to define the usable area of the page. The position of StreamFrame objects and components bound to the PageTemplate is relative to the top left corner of the usable area and cannot extend beyond the bottom right corner.

marginLeft indicates the distance, in the report's current *metric* units, between the left edge of the page and the left edge of the usable area.

When using multiple PageTemplate objects, you can position items differently just by changing the margins. For example, you may use different left and right pages that have increased margins on the inside edge (the gutter space) for binding.

marginRight

Topic group Related topics

The space between the right edge of the page and the usable area of the PageTemplate.

Property of

PageTemplate

Description

Use *marginRight* in conjunction with the PageTemplate's other margin- properties to define the usable area of the page. The position of StreamFrame objects and components bound to the PageTemplate is relative to the top left corner of the usable area and cannot extend beyond the bottom right corner.

marginRight indicates the distance, in the report's current *metric* units, between the right edge of the page and the right edge of the usable area.

When using multiple PageTemplate objects, you can position items differently just by changing the margins. For example, you may use different left and right pages that have increased margins on the inside edge (the gutter space) for binding.

marginTop

Topic group Related topics

The space between the top of the page and the usable area of the PageTemplate.

Property of

PageTemplate

Description

Use *marginTop* in conjunction with the PageTemplate's other margin- properties to define the usable area of the page. The position of StreamFrame objects and components bound to the PageTemplate is relative to the top left corner of the usable area and cannot extend beyond the bottom right corner.

marginTop indicates the distance, in the report's current *metric* units, between the top of the page and the top of the usable area.

When using multiple PageTemplate objects, you can position items differently just by changing the margins. For example, you may use different left and right pages that have increased margins on the inside edge (the gutter space) for binding.

marginVertical

Topic group Related topics

The vertical margin between the edge of the object and its contents. For use in Reports only.

Property of

Editor, Text, StreamFrame

Description

An object's vertical margin is the same on both the top and bottom. All rendering in the object, whether it's text in a Text component or bands inside a StreamFrame object, is relative to the vertical margin on the top and restricted by the vertical margin on the bottom.

marginVertical is measured in the form or report's current *metric* units.

maxRows

Topic group Related topics

The maximum number of rows a StreamSource will provide to a StreamFrame on each page.

Property of

StreamSource

Description

Use *maxRows* to limit the number of rows displayed in each StreamFrame. For example, on a page with a single StreamFrame, if *maxRows* is 10, only 10 rows maximum will displayed in the StreamFrame on each page.

nextPageTemplate

Topic group Related topics

The PageTemplate object that is used for the following page.

Property of

PageTemplate

Description

Because a report may have multiple PageTemplate objects, the *firstPageTemplate* property is used to identify the PageTemplate that the report should render as its first page.

Once the first PageTemplate has been chosen, each PageTemplate object has a *nextPageTemplate* property that identifies the page to render next.

For a report that uses the same page over and over, the same PageTemplate object is used for both the report's *firstPageTemplate* property and that PageTemplate's own *nextPageTemplate* property.

You can create a different introduction or cover page for a report by specifying the cover page as the report's *firstPageTemplate* property, and then set the cover page's *nextPageTemplate* property to the PageTemplate for the body pages.

To alternate left and right pages, set the *nextPageTemplate* of the left page to the right page, and vice versa. Then specify the page on the right as the report's *firstPageTemplate*.

onPage

Topic group Example

After the page has finished rendering.

Parameters

none

Property of

Report

Description

onPage fires after each page has finished rendering, including the last page. By that time, it's too late to do anything to the page, but you can take actions for the next page.

In an *onPage* event handler, the report's *reportPage* property indicates the page that has just finished rendering.

onRender

Topic group Related topics Example

After the component is rendered.

Parameters

none

Property of

All visual components in a report, Band, StreamFrame

Description

onRender fires for visual components only when they are in a report. It is fired every time the object is rendered, after it has finished rendering. For a component in a detail band, that means for every row in the rowset.

You can use the *onRender* event to reset the component to its default state after changing it in its *canRender* event.

onRender also fires for the Band objects, after all the objects in that band have been rendered (for a detail band, after every row); and StreamFrame objects, after that StreamFrame has been rendered, on each page.

output

Topic group Related topics

Designates the target medium for the report output.

Property of

Report

Description

Set the report's *output* property to designate how you want the report to be rendered. *output* may contain one of the following values:

Value	Target
0	Window

1	Printer
2	Printer file
3	Default
4	HTML file
5	CGI Response

The Default output is to a window. When rendering to a window, only one page at a time is rendered.

If you designate Printer file, HTML file, or CGI Response, the report's *outputFilename* property must be set to the name of the target file.

The CGI Response option allows HTML report output to be streamed directly out to Web servers. Reports which have been compiled and built into executables (.EXEs) can be called and run directly from a URL typed into the Web browser. In the design environment, the CGI Response option behaves identically to the HTML File output option in that the report output is written to *outputFilename.* In the runtime environment, *outputFilename* is ignored and the HTML output is automatically streamed to *stdout*.

outputFilename

Topic group Related topics

The name of the target output file.

Property of

Report

Description

If a report's *output* property is set to Printer file, HTML file, or CGI Response, the *outputFilename* property must be set to the target file.

When the *output* property is CGI Response, *outputFilename* is ignored in the runtime environment. It is only used when the report is run in the design environment.

The file will contain the output from the report. If the file already exists, it will be overwritten.

preRender

Topic group Related topics

Before the band begins rendering.

Parameters

none

Property of

Band

Description

A band's *preRender* event fires before the band starts rendering. Note that unlike *canRender*, you cannot prevent the band from rendering in the *preRender* event handler.

printer

Topic group Related topics

An object that describes various printer output options.

Property of

Report

Description

A *printer* object contains properties for the following printer options:

Property	Default	Description
color	Monochrome	Whether the output should be in color/grayscale or plain monochrome (0=Default, 1=Monochrome, 2=Color)
copies	1	Number of copies
duplex	None	Whether to print in duplex, and in which orientation (0=Default, 1=None, 2=Vertical, 3=Horizontal)
orientation	Portrait	The orientation of the output (0=Default, 1=Portrait, 2=Landscape)
paperSize	printer-dependent	The size of the paper to use
paperSource	printer-dependent	The paper tray or bin to use
printerName		The name of the target printer (blank for default printer)
printerSource	dBASE Plus default	Which printerName to use (0=dBASE Plus default, 1=Windows default, 2=Specific)
resolution	High	Graphics resolution (0=Default, 1=Draft, 2=Low, 3=Medium, 4=High)
trueTypeFonts	Depends on the currently specified default printer	How to handle TrueType fonts (0=Default, 1=Bitmap, 2=Download, 3=Substitute, 4=Outline)

There is only one method associated with the printer object.

Method	Parameters	Description
choosePrinter( )	<title expC> <expL>	Displays Print or Print Setup dialog to allow user to choose a printer and set printing options

Each Report object has its own *printer* object which controls the report output. The global *_app* object also has a *printer* object that represents the default printer.

The ***color*** property determines whether the default output should be color, greyscale, plain monochrome, or default (the printer driver's default). If you do not have a color printer attached, colors will be evaluated and printed to greyscale.

The ***copies*** property lets you set the number of copies to print.

The ***duplex*** property is used to determine whether to print in duplex mode for printers that support this option.

The ***orientation*** property is used to determine whether a report should print in portrait, or landscape mode.

The ***paperSize*** property allows you to set the paper size for the report. It uses a numeric scheme that varies based on the printer driver. Some printers support paper sizes that others do not.

For printers that support this option, the ***paperSource*** property allows you to determine which tray, or bin, the paper will come from. You should be able to change the *paperSource* on the fly, which might prove useful should you wish to print company letterhead on a report's cover page only.

The ***printerName*** property stores the name of the printer. If a report's *printer* object's *printerName* property is blank, the printer specified in *_app.printer.printerName* is used (all other properties in the report's *printer* object are applied). If *_app.printer.printerName* is blank, the default Windows printer is used. In most cases you should not

set this property. Among other things, the *printerName* property will be saved in the report's source code, which might cause problems should you attempt to render the report on a computer that does not have this printer available.

The ***printerSource*** property determines which *printerName* should be used. For the reason stated above (see *printerName*), using the *choosePrinter*( ) method is recommended.

The ***resolution*** property allows you to set the resolution for your report's output. This might prove useful if you were using a dot matrix printer to test your report. Since high resolution output on these printers is usually tediously slow, you could drop the resolution to low when testing, and only use high resolution for the final output.

The ***trueTypeFonts*** property can be used to determine how the printer will handle TrueType fonts. For the most part it is probably safest to leave this alone.

Call the ***choosePrinter***( ) method to allow the user to choose a printer and set printing options. Each *printer* object's *choosePrinter*( ) method changes the printer settings for that object.

Calling the report object's *choosePrinter*( ) method

```
report.printer's choosePrinter( )
```

allows you to set the printer for the current report. This could be useful in a networked office where several printers are available. Each user could designate a printer where their report should be sent.

Calling the _app object's *choosePrinter*( ) method

```
_app.printer.choosePrinter( )
```

sets the printer for your whole application; the same as the CHOOSEPRINTER( ) function.

paperSize

Topic group

Set's the paper size to use when printing.

Syntax

<oRef>.paperSize = nPaperSize

<oRef>

A reference to the printer object whose *paperSize* you wish to set.

Property of:

printer class

Description

When setting the paperSize only settings supported by the printer being used should be set. Here is a list of the printer options by number:

1. Letter, 8 1/2 x 11 in.
2. Letter Small, 8 1/2 x 11 in.
3. Tabloid, 11 x 17 in.
4. Ledger, 17 x 11 in.
5. Legal, 8 1/2 x 14 in.
6. Statement, 5 1/2 x 8 1/2 in.
7. Executive, 7 1/2 x 10 1/2 in.
8. A3, 297 x 420 mm
9. A4, 210 x 297 mm

10. A4 Small, 210 x 297 mm
11. A5, 148 x 210 mm
12. B4, 250 x 354 mm
13. B5, 182 x 257 mm
14. Folio, 8 1/2 x 13 in.
15. Quarto, 215 x 275 mm
16. 10 x 14 in.
17. 11 x 17 in.
18. Note, 8 1/2 x 11 in.
19. Envelope #9, 3 7/8 x 8 7/8 in.
20. Envelope #10, 4 1/8 x 9 1/2 in.
21. Envelope #11, 4 1/2 x 10 3/8 in.
22. Envelope #12, 4 1/2 x 11 in.
23. Envelope #14, 5 x 11 1/2 in.
24. C size sheet
25. D size sheet
26. E size sheet
27. Envelope DL, 110 x 220 mm
28. Envelope C3, 324 x 458 mm
29. Envelope C4, 229 x 324 mm
30. Envelope C5, 162 x 229 mm
31. Envelope C6, 114 x 162 mm
32. Envelope C65, 114 x 229 mm
33. Envelope B4, 250 x 353 mm
34. Envelope B5, 176 x 250 mm
35. Envelope B6, 176 x 125 mm
36. Envelope, 110 x 230 mm
37. Envelope Monarch, 3 7/8 x 7 1/2 in.
38. Envelope, 3 5/8 x 6 1/2 in.
39. U.S. Standard Fanfold, 14 7/8 x 11 in.
40. German Standard Fanfold, 8 1/2 x 12 in.
41. German Legal Fanfold, 8 1/2 x 13 in.

256. User-defined

render( )

Topic group Related topics Example

Renders the reports to the designated target.

Syntax

<oRef>.*render*( )

<oRef>

An object reference to the report you want to render.

Property of

Report

Description

Call a Report object's *render*( ) method to generate the report. The output of the report goes to the target designated by the report's *output* property.

The report engine renders the report internally and outputs the results starting with the page designated by the *startPage* property. When rendering to a window (the default), rendering stops once a page is output; the window only displays one page at a time. Rendering to a printer or file renders multiple pages. It continues to render all the objects on each page until it exhausts all StreamSource objects, or it has finished rendering the page designated by the *endPage* property.

renderOffset

Topic group Related topics Example

The offset of the bottom of the band from the top of the current stream frame.

Property of

Band

Description

renderOffset is a read-only property that contains the position of the bottom of the band that was just rendered, measured from the top of the stream frame, in the report's current *metric* units. It is updated just before the band's *onRender* event fires.

A common use of *renderOffset* is to prevent a band from being split between two pages when using *expandable* bands—that is, bands that can vary in height. The *renderOffset* in checked the *onRender* event. If the offset is too close to the bottom, less than a predetermined minimum size, the stream source's *beginNewFrame*( ) method is called to force the output to the next stream frame. For fixed height bands, the stream source's *maxRows* property would used instead.

reportGroup

Topic group

A Group object for the report as a whole.

Property of

Report

Description

A Report object automatically has a Group object assigned to its *reportGroup* property. The *groupBy* property of this Group object is an empty string.

Use the *reportGroup* to calculate aggregates for the entire report, for example, a grand total; and for items in a report introduction or summary.

A *reportGroup* has a *headerBand*, a *footerBand*, and aggregate methods, just like any other Group object. Because the *parent* of the *reportGroup* is the Report object instead of a StreamSource or Group object, the object reference path to data is slightly different for components in the *reportGroup*.

reportPage

Topic group Example

The current page number being rendered.

Property of

Report

Description

The *reportPage* property contains the number of the page that is being rendered.

During an *onPage* event, the *reportPage* is the page that has just finished rendering.

reportViewer

Topic group

A reference to the ReportViewer object that instantiated the report.

Property of

Report

Description

Use the *reportViewer* property to reference the ReportViewer object that instantiated the report. You may access the form that contains the ReportViewer through the ReportViewer object's *form* property.

If the report was not instantiated through a ReportViewer object, *reportViewer* is *null*.

rotate

Topic group

The orientation of an object, in 90-degree increments.

Property of

Text

Description

To rotate the text inside an Text component, set its *rotate* property to one of the following values:

Value	Clockwise rotation
0	none
1	90 degrees
2	180 degrees
3	270 degrees

startPage

Topic group Related topics

The first page number to output.

Property of

Report

Description

When rendering to a window (the default), only one page, the *startPage*, is rendered at a time. When rendering to a printer or file, pages are rendered from the *startPage* to the *endPage*.

By default, *startPage* is 1, which means that all the pages that are rendered are output. Set *startPage* to a number greater than 1 to delay the beginning of the output.

The report engine must still render each page until it gets to the *startPage* because it dynamically paginates the report. The position of a row in a report may change whenever someone changes the table, so use caution when using *startPage* to display segments of a report.

streamFrame

Topic group

The StreamFrame object in which the band is currently rendering.

Property of

Band

Description

The read-only *streamFrame* property contains a reference to the StreamFrame object in which the band is currently rendering. In addition to being a direct reference to that object, it is particularly useful when you have more than one stream frame, and want to take different actions dependent on which stream frame is being rendered.

streamSource

Topic group

The StreamSource object that contains objects to render in the StreamFrame.

Property of

streamFrame

Description

A StreamFrame object's *streamSource* property identifies the StreamSource object that supplies the StreamFrame object's data stream.

If multiple StreamFrame objects have the same *streamSource* property, that StreamSource will stream data to those StreamFrame objects in series, in the order in which the StreamFrame objects were created (the same order as they are listed in the class definition in the .REP file).

If multiple StreamFrame objects have different *streamSource* properties, each StreamFrame will be filled from its own StreamSource object (in the same order as above) until all the StreamFrame objects in the page are filled. If a particular StreamFrame object's StreamSource is exhausted, it is no longer filled. When all StreamSource objects are exhausted, all the objects on that last PageTemplate are rendered, and the report is done.

suppressIfBlank

Topic group Related topics

Specifies whether an object is suppressed, or not rendered, if it is blank. For use in Reports only.

Property of

Text

Description

suppressIfBlank has an effect only for visual components that are in a report. If *true*, it suppresses the rendering of the component if its display value is blank.

For example, suppose you're printing two-line addresses with the city underneath. If the second address line is blank, you don't want it to occupy any space. By setting that component's *suppressIfBlank* property to *true*, if it's blank, nothing gets rendered and all the components below it that have their *fixed* properties set to *false* are moved up to fill the space.

Components are rendered in their z-order, the order in which they are defined in the report. *suppressIfBlank* does not affect the position of components that have already been rendered. The z-order of the components in the report should match their top-to-bottom order; otherwise *suppressIfBlank* will have no effect.

By default *suppressIfBlank* is *false*. You can also use the component's *canRender* event to suppress rendering.

suppressIfDuplicate

Topic group Related topics

Specifies whether an object is suppressed, or not rendered, if its value is the same as the previous time it was rendered. For use in Reports only.

Property of

Text

Description

suppressIfDuplicate has an effect only for visual components that are in a report. If *true*, it suppresses the rendering of the component if its display value is the same as the previous time it was rendered in the same report, even if the previous time was in another group in the report.

Use *suppressIfDuplicate* to eliminate duplicating the same data value over and over again for multiple rows in a report. For example, you may have information sorted by date. With *suppressIfDuplicate* set to *true*, each date will be rendered only once.

By default *suppressIfDuplicate* is *false*. You can also use the component's *canRender* event to suppress rendering.

title

Topic group

The title of a report.

Property of

Report

Description

The *title* property contains the title of the report. It is displayed in the title bar of the report preview window.

tracking

Topic group Related topics

The amount of extra space between characters. For use in Reports only.

Property of

Text

Description

tracking adds extra space between characters within an Text component. By default, it's zero, which means no extra spacing.

You can set *tracking* to a non-zero value to add extra space between characters.

trackJustifyThreshold

Topic group Related topics

The maximum amount of added space allowed between words in a fully justified line. Exceeding that amount switches to character tracking. For use in Reports only.

Property of

Text

Description

trackJustifyThreshold sets a threshold for the amount of extra space between words that can be added to try to justify the line. If a line requires more than the threshold amount, the line is justified by adding space between each character in the line, in addition to the maximum space between each word.

If a line contains only one word and *trackJustifyThreshold* is non-zero, the word will be fully justified with character tracking, unless it is on the last line of text. The last line of text is never justified.

An Text component's *alignHorizontal* property must be set to Justify in order for *trackJustifyThreshold* to have any effect.

variableHeight

Topic group Related topics

Whether an object's *height* can increase automatically to accommodate its contents. For use in Reports only.

Property of

Text

Description

Set *variableHeight* to *true* so that an object can grow to accommodate its contents. If an object's *height* is not large enough to display everything, it is increased. *variableHeight* does not shrink objects to fit their contents.

If the object is in a Band object in a report and it grows, it might push down other objects in the band if those objects have their *fixed* property set to *false*.

By default *variableHeight* is *false*.

verticalJustifyLimit

Topic group Related topics

The maximum amount of added space allowed between lines in a vertically justified object. Exceeding that amount makes the text top-aligned instead. For use in Reports only.

Property of

Text

Description

verticalJustifyLimit sets the maximum amount of extra space between lines that can be added to try to vertically justify the lines in an object. If the maximum amount does not justify the lines, *dBASE Plus* gives up and makes the text top-aligned instead.

An Text component's *alignVertical* property must be set to Justify in order for *verticalJustifyLimit* to have any effect.

Files and Operating System

File commands and functions

Topic group

dBASE Plus supports equivalent file commands and functions for all the methods in the File class, which can be organized into the following categories:

File utility commands
File information functions

Functions that provide byte-level access to files, sometimes referred to as low-level file functions

The low-level file functions are maintained for compatibility. To read and write to files, you should use a File object, which better encapsulates direct file access. In contrast, the file utility commands and file information functions are easier to use, because they do not require the existence of a File object.

File utility commands

Topic group

The following commands have equivalent methods in the File class:

Command	File class method
COPY FILE	*copy*( )
DELETE FILE	*delete*( )
ERASE	*delete*( )
RENAME	*rename*( )

These commands are described separately to document their syntax. Otherwise, they perform identically to their equivalent method.

File information functions

Topic group

The following file information functions are usually used instead of their equivalent methods in the File class:

Function	File class method
FACCESSDATE( )	accessDate( )
FCREATEDATE( )	createDate( )
FCREATETIME( )	createTime( )
FDATE( )	date( )
FILE( )	exists( )
FSHORTNAME( )	shortName( )
FSIZE( )	size( )
FTIME( )	time( )

These functions are not described separately (except for FILE( ), because its name is not based on the name of its equivalent method). The syntax of a file information function is identical to the syntax of the equivalent method, except that as a function, no reference to a File object is needed, which makes the function more convenient to use. For example, these two statements are equivalent

```
nSize = fsize( cFile ) // Get size of file named in variable cFile
nSize = new File( ).size( cFile ) // Get size of file named in variable cFile
```

Low-level file functions

Topic group

The following low-level file functions are equivalent to the following methods in the File class:

Function	File class method
FCLOSE( )	*close*( )
FCREATE( )	*create*( )
FEOF( )	*eof*( )
FERROR( )	*error*( )
FFLUSH( )	*flush*( )
FGETS( )	*gets*( ) and *readln*( )
FOPEN( )	*open*( )
FPUTS( )	*puts*( ) and *writeln*( )
FREAD( )	*read*( )
FSEEK( )	*seek*( )
FWRITE( )	*write*( )

These functions are not described separately. While a File object automatically maintains its file handle in its *handle* property, low-level file functions must explicitly specify a file handle, with the exception of FERROR( ), which does not act on a specific file. The FCREATE( ) and FOPEN( ) functions take the same parameters as the *create*( ) and *open*( ) methods, and return the file handle.

The other functions use the file handle as their first parameter and all other parameters (if any) following it The parameters following the file handle in the function are identical to the parameters to the equivalent method, and the functions return the same values as the methods.

Compare the examples for *exists*( ) and FILE( ) to see the difference between using a File object and low-level file functions.

Dynamic External Objects (DEO)

Topic group Related topics

Dynamic External Objects is a unique technology that allows not just users, but applications, to share classes across a network (and soon, across the Web). Instead of linking your forms, programs, classes and reports into a single executable that has to be manually installed on each workstation, you deploy a shell - a simple *dBASE Plus* executable that calls an initial form, or provides a starting menu from which you can access your forms and other *dBASE Plus* objects. The shell executable can be as simple a program as:

```
do startup.prg
```

where "startup.prg" can be a different ".pro" object in each directory from which you launch the application, or a program that builds a dynamic, context-sensitive menu on-the-fly.

Dynamic Objects can be visual, or they can be classes containing just "business rules", that process and post transactions, or save and retrieve data. Each of these objects may be shared across your network by all users, and all applications that call them.

For example, you may have a customer form that's used in your Customer Tracking application, but may also be used by your Contact Management program as well as your Accounts Receivable module. Assume you want to change a few fields on the form or add a verification or calculation routine. No problem, just compile the new form and use the Windows Explorer to

drag it to the appropriate folder on your server. Every user and application is updated immediately.

Benefits of using Dynamic External Objects:

Updating objects requires only a simple drag-and-drop. No registration, no interface files, no Application Server required. Updating has never before been this easy.

Although the objects sit on your network server, they run on the workstation, reducing the load on the Server dramatically and making efficient use of all that local processing power sitting out on your network.

The same (non-visual) objects may be shared by both your LAN and your Web site. Dynamic External Objects are very small and load incredibly fast. They rarely exceed 140K in size and usually run less than 100K.

And, most remarkable of all, this is one of the only Object Models that supports full inheritance. You can't inherit from an ActiveX/OCX object. You can inherit Java objects in CORBA, but it's so difficult it's rarely attempted. With *dBASE Plus*, inheriting External Objects is a piece of cake:

Change the layout of a superclass form and every form in every application inherits those changes the next time they're called.

Renamed your company? Changed your logo? Drag and drop the new .cfo file to the Server and the update is finished.

Implementing Dynamic Objects

- Compile your source code as you would normally. DEO uses compiled code. In *dBASE Plus*, compiled code can be recognized because its file extension ends in "o". .Rep, compiled, becomes .Reo, .Wfm becomes .Wfo, etc.
- Build (create executable) only the main launching form, or use a pre-built generic launching form.
- Copy your objects to the server.

Done !

Like Source Aliasing, DEO has a mechanism for finding libraries of objects, making it much easier to share them across the network and across applications. This mechanism is based on an optional search list that's created using easy text changes in your application's .ini file.

***dBASE Plus* searches objects as follows:**

1. It looks in the "home" folder from which the application was launched.
2. It looks in the .ini file to see if there's a series of search paths specified. It checks all the paths in the list looking for the object file requested by the application.
3. It looks inside the application's .exe file, the way Visual dBASE did in the past.

Let's assume you have a library in which you plan to store your shared objects. Let's also assume the application is called "Myprog.exe" and runs from the c:\project1 folder.

In Myprog.ini, we might add the following statements:

```
[ObjectPath]
objPath0=f:\mainlib
objPath1=h:\project1\images
objPath2=f:\myWeb
```

Your code looks something like the following:

```
set procedure to postingLib.cc additive
```

dBASE Plus will first look in c:\project1 (the home directory).

If that fails, *dBASE Plus* will look in f:\mainlib. If it finds postingLib.co, it will load that version. If not, it looks in each of the remaining paths on the list until it finds a copy of the object file.

If that fails, *dBASE Plus* will look inside MyProg.exe.

Tips

You'll have to experiment with DEO to discover the best approach for the way you write and deploy applications. However, here are some interesting subtleties you might leverage to your benefit:

Unanticipated updates: Assume you already shipped a *dBASE Plus* application as a full-blown executable. Now you want to make a change to one module. No problem, just copy the object file to the home directory of the application and it'll be used instead of the one built in to the executable. You don't need to redeploy the full application the way you do in most other application development products. Just the changed object.

Reports: You can deploy reports and add them to their applications by designing a report menu that checks the disk for files with an .reo extension. Let the menu build itself from the file list. Here we have true dynamic objects - the application doesn't even know they exist until runtime. DEO supports real-time dynamic applications.

Tech Support: Want to try out some code or deploy a fix to a customer site or a remote branch office? No problem, just FTP the object file to the remote server and the update is complete.

Remote Applications: If you have VPN support (or any method of mapping an Internet connection to a drive letter), you can run *dBASE Plus* DEO applications remotely over the Internet. A future version of *dBASE Plus* will include resolution of URLs and IP addresses so you can access remote objects directly through TCP/IP without middleware support.

Distributed Objects: Objects can be in a single folder on your server, in various folders around your network, or duplicated in up to ten folders for fail-over. If one of your servers is down, and an object is unavailable, *dBASE Plus* will search the next locations on the list until it finds one it can load. Objects can be located anywhere they can be found by the workstation.

Source Aliasing

Topic group

What is Source Aliasing?

Source Aliasing is a feature in *dBASE Plus* that provides true source-code portability by referencing files indirectly - through an Alias. Just as the BDE allows you to define an Alias to represent a database or a collection of tables, Source Aliases let you define locations for your various files without using explicit paths - which often differ from machine to machine.

For example, if you're using seeker.cc in a *dBASE Plus* application, you're likely to have code similar to the following:

```
set procedure to "c:\program files\dBASE\Plus\Samples
\seeker.cc" additive
```

If you run this code on another machine, whose application drive is not "c:", it will crash.

You can avoid portability problems like the example above (as well as save a lot of typing) by using a Source Alias in place of explicit paths:

```
set procedure to :MainLib:seeker.cc additive
```

Whenever *dBASE Plus* sees ":MainLib:", it automatically substitutes the path assigned to this Alias. To run the same code on another computer or drive, simply set up the "MainLib" Alias to point to the appropriate folder at the new location. No source code changes are required.

There are other major benefits to Source Aliasing.

You can run applications from within *dBASE Plus* regardless of their location and current folder. Every application always finds all of its parts.

You can build well-organized, reliable libraries of object source that can be accessed across many projects without dealing with complicated and changing paths. You may, for example, want to:

- Build a MAIN alias that represents a folder in which you store globally shared classes.
- Use an IMAGES alias to point to a location containing all your reusable bitmaps, .gifs and .jpgs.
- Build a PROJECT1 alias for classes and code associated only with one specific project.

If you're careful to always use Source Aliases, your libraries will be shared with ease, and portable enough to be shared across a network by other developers and users.

Using Source Aliases

To create a new Source Alias, go to Properties|Desktop Properties menu option and click on the "Source Aliases" tab. *dBASE Plus* can support an unlimited number of Source Aliases.

There are at least three ways to use Source Aliases in *dBASE Plus.*.

1. When hand-coding, always use an alias preceded and followed by a colon:

```
set procedure to :newSA:my.wfm additive
dataSource := "FILENAME :newSA:NewButton.bmp"
upBitMap := ":newSA:OKButton.bmp"
do :newSA:Main.prg
```

2. When setting properties (such as Bitmaps) in the Inspector, always add the Source Alias to a filename.
3. *dBASE Plus* may add Source-Aliases automatically. In many cases, *dBASE Plus* will substitute the correct Source Alias whenever you select a file from an Open File dialog, drag-and-drop a file from the Navigator, or type in an explicit path.

Source Alias information is stored in the PLUS.ini file. As a result, you need to add the Source Alias to any *dBASE Plus* installation that will run your code. You can add Aliases programmatically by modifying the PLUS.ini file.

You can retrieve the paths associated with Source Aliases through the sourceAliases property of the main application object. For example:

```
? _app.sourceAliases["ResView"]
```

returns

```
c:\program files\dBASE\Plus\Bin\ResView (or c:\program files (x86)\dBASE\Plus\Bin\ResView)
```

Be careful! _app.sourceAliases is an Associative Array and is, therefore, case-sensitive. Capitalization must match the Alias name you set up in Desktop Properties.

Important Note: Source Aliasing works only in the *dBASE Plus* design environment or when running programs from within the *dBASE Plus* shell. It is not a runtime feature. To access files indirectly in deployed applications, use DEO (Dynamic External Objects) instead of Source Aliasing.

class File

Topic group Example

An object that provides byte-level access to files and contains various file directory methods.

Syntax

[<oRef> =] new File( )

<oRef>

A variable or property in which to store a reference to the newly created File object.

Properties

The following tables list the properties and methods of the File class. (No events are associated with this class.)

Property	Default	Description
baseClassName	FILE	Identifies the object as an instance of the File class
className	(FILE)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
handle	–1	Operating system file handle
path		Full path and file name for open file
position	0	Current position of file pointer, relative to the start of the file

Method	Parameters	Description
accessDate( )	<filename expC>	Returns the last date a file was opened
close( )		Closes the currently open file
copy( )	<filename expC> , <new name expC>	Makes a copy of the specified file
create( )	<filename expC> [,<access rights>]	Creates a new file with optional access attributes
createDate( )	<filename expC>	Returns the date when the file was created
createTime( )	<filename expC>	Returns the time a file was created as a string
date( )	<filename expC>	Returns the date the file was last modified
delete( )	<filename expC>	Deletes the specified file
eof( )		Returns true or false indicating if the file pointer is positioned past the end of the currently open file
error( )		Returns a number indicating the last error encountered
exists( )	<filename expC>	Returns true or false to indicate whether the specified disk file exists
flush( )		Writes current data in the file buffer to disk and keeps file open
gets( )	[<chars read expN>] [, <eol expC>]	Reads and returns a line from a file, leaving the file pointer at the beginning of the next line. Same as readln( )
open( )	<filename expC> [,<access rights>]	Opens an existing file with optional access attributes
puts( )	<input string expC> [, <max chars expN> [, <eol expC>]	Writes a character string and end-of-line character(s) to a file. Same as writeln( )
read( )	<characters expN>	Reads and returns the specified number of characters from the file starting at the current file pointer position. Leaves the file pointer at the character following the last one read.
readln( )	[<chars read expN>] [, <eol expC>]	Reads and returns a line from a file, leaving the file pointer at the beginning of the next line. Same as gets( ).
rename( )	<filename expC> , <new name expC>	Changes the name of the specified file to a new name
seek( )	<offset expN> [, 0 | 1 | 2]	Moves the file pointer the specified number of bytes within a file, optionally allowing the movement to be from the beginning (0), end (2), or current file position (1)
shortName( )	<filename expC>	Returns the short (8.3) name for a file
size( )	<filename expC>	Returns the number of bytes in the specified file

time( )	<filename expC>	Returns the time the file was last modified as a string
write( )	<expC> [, <max chars expN>]	Writes the specified string into the file at the current file position, overwriting any existing data and leaving the file pointer at the character after the last character written
writeln( )	<input string expC> [, <max chars expN> [, <eol expC>]	Writes a character string and end-of-line character(s) to a file. Same as puts( ).

Description

Use a File object for direct byte-level access to files. Once you create a new File object, you can *open*( ) an existing file or *create*( ) a new one. Be sure to *close*( ) the file when you are done. A File object may access only one file at a time, but after closing a file, you may open or create another.

To communicate directly with a Web server, use the File object's *open*( ) method to access "StdIn" or "StdOut".

To open StdIn use:

```
fIn = new File()
fIn.open( "StdIn", "RA")
```

To open StdOut use:

```
fOut = new File()
fOut.open( "StdOut", "RA")
```

When reading or writing to a binary file, be sure to specify the "B" binary access specifier. Without it, the file is treated as a text file; if the current language driver is a multi-byte language, each character in the file may be one or two bytes. Binary access ensures that each byte is read and written without translation.

File objects also contain information and utility methods for file directories, such as returning the size of a file or changing a file name. If you intend to call multiple methods, you can create and reuse a File object. For example,

```
ff = new File( )
? ff.size( "Plus_en.hlp" )
? ff.accessDate( "Plus_en.hlp" )
```

Or you can create a File object for a WITH block. For example,

```
with new File( )
? size( "Plus_en.hlp" )
? accessDate( "Plus_en.hlp" )
endwith
```

For a single call, you can create and use the File object in the same statement:

```
? new File( ).size( "Plus_en.hlp" )
```

However, unless you happen to have a File object handy, it's easier to use the equivalent built-in function or command to get the file information or perform the file operation:

```
? fsize( "Plus_en.hlp" )
? faccessdate( "Plus_en.hlp" )
```

Topic group Related topics

Executes a program or operating system command from within *dBASE Plus*.

Syntax

! <command>

<command>

A command recognized by your operating system.

Description

! is identical to RUN, except that a space is not required after the ! symbol, while a space is required after the word RUN. See RUN for details.

accessDate( )

Topic group Related topics Example

Returns the last date a file was opened.

Syntax

<oRef>.*accessDate*(<filename expC>)

<oRef>

A reference to a File object.

<filename expC>

The name of the file to check. Wildcard characters are not allowed; you must specify the actual file name.

If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory, then in the search path(s) you specified with SET PATH, if any. If you specify a file without including its extension, *dBASE Plus* assumes no extension. If the named file cannot be found, an exception occurs.

Property of

File

Description

accessDate( ) checks the file specified by <filename expC> and returns the date that the file was last opened by the operating system for reading or writing.

To get the date the file was last modified, use *date*( ). For the date the file was created, use *createDate*( ).

CD

Topic group Related topics

Changes the current drive or directory.

Syntax

CD [<path>]

<path>

The new drive and/or path. Quotes (single or double) are required if the path contains spaces or other special characters; optional otherwise.

Description

Use CD to change the current working directory in *dBASE Plus* to any valid drive and path. If you're unsure whether a drive is valid, use VALIDDRIVE( ) before issuing CD. The current directory appears in the Navigator.

CD supports the Universal Naming Convention (UNC), which starts with double backslashes for the resource name, for example:

```
\\MyServer\MyVolume\MyDir\MySubdir
```

CD without the option <path> displays the current drive and directory path in the result pane of the Command window. To get the current directory, use SET("DIRECTORY").

Another way to access files on different directories is with the command SET PATH. You can specify one or more search paths, and *dBASE Plus* uses these paths to locate files not on the current directory. Use SET PATH when an application's files are in several directories.

CD works like SET DIRECTORY, except SET DIRECTORY TO (with no argument) returns you to the HOME( ) directory, instead of displaying the current directory.

close( )

Topic group Related topics Example

Closes a file previously opened with *create*( ) or *open*( ).

Syntax

<oRef>.*close*( )

<oRef>

A reference to the File object that created or opened the file.

Property of

File

Description

close( ) closes a file you've opened with *create*( ) or *open*( ). *close*( ) returns *true* if it's able to close the file. If the file is no longer available (for example, the file was on a floppy disk that has been removed) and there is data in the buffer that has not yet been written to disk, *close*( ) returns *false*.

Always close the file when you're done with it.

To save the file to disk without closing it, use *flush*( ).

copy( )

Topic group Related topics Example

Duplicates a specified file.

Syntax

<oRef>.*copy*(<filename expC>, <new name expC>)

<oRef>

A reference to a File object.

<filename expC>

Identifies the file to duplicate (also known as the source file). <filename expC> may be a file name skeleton with wildcard characters. In that case, *dBASE Plus* displays a dialog box in which you select the file to duplicate.

If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory, then in the search path(s) you specified with SET PATH, if any. If you specify a file without including its extension, *dBASE Plus* assumes no extension. If the named file cannot be found, an exception occurs.

<new name expC>

Identifies the target file that will be created or overwritten by *copy*( ). <new name expC> may be a filename skeleton with wildcard characters. In that case, *dBASE Plus* displays a dialog box in which you specify the name of the target file and its directory.

Property of

File

Description

copy( ) lets you duplicate an existing file at the operating system level. *copy*( ) duplicates a single file of any type.

When running a PLUS.exe, *copy*( ) first looks for <filename expC> in the internal file system of the .EXE file. Any path in <filename expC> is ignored. If the named file is found in the .EXE, that file is copied. If the file is not found, then *dBASE Plus* searches for the file on disk. This lets you package static files, like empty tables, inside the .EXE during the build process and extract them when needed. You cannot copy files into the .EXE

If SET SAFETY is ON and a file exists with the same name as the target file, *dBASE Plus* displays a dialog box asking if you want to overwrite the file. If SET SAFETY is OFF, any existing file with the same name is overwritten without warning.

copy( ) does not automatically copy the auxiliary files associated with table files, such as indexes and memo files. For example, it does not copy the MDX or DBT file associated with a DBF file. When copying tables, use the Database object's *copyTable*( ) method.

You cannot *copy*( ) a file that has been opened for writing with the *open*( ) or *create* ( ) methods; it must be closed first.

COPY FILE

Topic group Related topics Example

Duplicates a specified file.

Syntax

COPY FILE <filename> TO <new name>

Description

COPY FILE is identical to the File object's *copy*( ) method, except that as a command, the file name arguments are treated as names, not character expressions. They do not require quotes unless they contain spaces or other special characters. If the name is in a variable, you must use the indirection or macro operators.

See *copy*( ) for details on the operation of the command.

create( )

Topic group Related topics Example

Creates and opens a specified file.

Syntax

<oRef>.*create*(<filename expC>[, <access expC>])

<oRef>

A reference to a File object.

<filename expC>

The name of the file to create and open. By default, *create*( ) creates the file in the current directory. To create the file in another directory, specify a full path name for <filename expC>.

<access expC>

The access level of the file to create, as shown in the following table. The access level string is not case-sensitive, and the characters in the string may be in any order. If omitted, the default is read and write text file. Append is a more restrictive version of write; the data is always added to the end of the file.

<access expC>	Access level
"R"	Read-only text file
"W"	Write-only text file
"A"	Append-only text file
"RW"	Read and write text file
"RA"	Read and append text file
"RB"	Read-only binary file
"WB"	Write-only binary file
"AB"	Append-only binary file
"RWB"	Read and write binary file
"RAB"	Read and append binary file

Property of

File

Description

Use *create*( ) to create a file with a name you specify, assign the file the level of access you specify, and open the file. If *dBASE Plus* can't create the file (for example, if the file is already open), an exception occurs.

SET SAFETY has no effect on *create*( ). If <filename expC> already exists, it is overwritten without warning. To see if a file with the same name already exists, use *exists*( ) before issuing *create*( ).

To use other File methods, such as *read*( ) and *write*( ), first open a file with *create*( ) or *open*( ).

When you open a file with *create*( ), the file is empty, so the file pointer is positioned at the first character in the file. Use *seek*( ) to position the file pointer before reading from or writing to a file.

createDate( )

Topic group Related topics Example

Returns the date a file was created.

Syntax

<oRef>.*createDate*(<filename expC>)

<oRef>

A reference to a File object.

<filename expC>

The name of the file to check. Wildcard characters are not allowed; you must specify the actual file name.

If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory, then in the search path(s) you specified with SET PATH, if any. If you specify a file without including its extension, *dBASE Plus* assumes no extension. If the named file cannot be found, an exception occurs.

Property of

File

Description

createDate( ) checks the file specified by <filename expC> and returns the date that the file was created.

To get the date the file was last modified, use *date*( ). For the date the file was last accessed, use *accessDate*( ). To get the time the file was created, use *createTime*( ).

createTime( )

Topic group Related topics Example

Returns the time a file was created.

Syntax

<oRef>.*createTime*(<filename expC>)

<oRef>

A reference to a File object.

<filename expC>

The name of the file to check. Wildcard characters are not allowed; you must specify the actual file name.

If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory, then in the search path(s) you specified with SET PATH, if any. If you specify a file without including its extension, *dBASE Plus* assumes no extension. If the named file cannot be found, an exception occurs.

Property of

File

Description

createTime( ) checks the file specified by <filename expC> and returns the time, as a character string, that the file was created.

To get the date the file was created, use *createDate*( ).

date( )

Topic group Related topics Example

Returns the date stamp for a file, the date the file was last modified.

Syntax

<oRef>.*date*(<filename expC>)

<oRef>

A reference to a File object.

<filename expC>

The name of the file to check. Wildcard characters are not allowed; you must specify the actual file name.

If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory, then in the search path(s) you specified with SET PATH, if any. If you specify a file without including its extension, *dBASE Plus* assumes no extension. If the named file cannot be found, an exception occurs.

Property of

File

Description

Use *date*( ) to determine the date on which the last change was made to a file on disk.

When you update a file, *dBASE Plus* changes the file's date stamp to the current operating system date when the file is written to disk. For example, when the user edits a DB table, *dBASE Plus* changes the date stamp on the table file when the file is closed. *date*( ) reads the date stamp and returns its current value.

To get the date the file was created, use *createDate*( ). For the date the file was last accessed, use *accessDate*( ). To get the time the file was last changed, use *time*( ).

delete( )

Topic group Related topics Example

Removes a file from a disk, optionally sending it to the Recycle Bin.

Syntax

<oRef>.*delete*(<filename expC> [, <recycle expL>])

<oRef>

A reference to a File object.

<filename expC>

Identifies the file to remove. <filename expC> may be a filename skeleton with wildcard characters. In that case, *dBASE Plus* displays a dialog box in which you select the file to duplicate.

If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory, then in the search path(s) you specified with SET PATH, if any. If you specify a file without including its extension, *dBASE Plus* assumes no extension. If the named file cannot be found, an exception occurs.

<recycle expL>

Whether to send the file to the Recycle Bin instead of deleting it. If omitted, the file is deleted.

Property of

File

Description

delete( ) deletes a file from a disk, or sends it to the Recycle Bin.

If <recycle expL> is *true*, then SET SAFETY determines whether a dialog appears to confirm sending the file to the Recycle Bin. If <recycle expL> is *false* or omitted, SET SAFETY has no effect on *delete*( ); the file is deleted without warning.

delete( ) does not automatically remove the auxiliary files associated with table files, such as indexes and memo files. For example, it does not delete the MDX or DBT files associated with a DBF file. When deleting tables, use the Database object's *dropTable*( ) method.

You cannot *delete*( ) a file that is open, including one opened with the *open*( ) or *create*( ) methods; it must be closed first.

DELETE FILE

Topic group Related topics Example

Removes a file from a disk.

Syntax

DELETE FILE <filename>

Description

DELETE FILE is similar to the File object's *delete*( ) method, except that as a command, the file name argument is treated as a name, not a character expression. It does not require quotes unless it contain spaces or other special characters. If the name is in a variable, you must use the indirection or macro operators. Also, DELETE FILE does not support sending a file to the Recycle Bin.

See *delete( )* for details on the operation of the command.

The ERASE command is identical to DELETE FILE.

DIR

Topic group Related topics Example

Performs a directory or table listing.

Syntax

DIR | DIRECTORY
[[LIKE] <drive/path/filename skeleton>]

[LIKE] <drive/path/filename skeleton>

Specifies a path and/or file specification to be used by DIR. The LIKE keyword is included for readability only; it has no effect on the command.

If omitted, the tables in the current directory or database are listed.

Description

DIR (or DIRECTORY) is a utility command that lets you perform a directory listing. The information provided on each file includes its short (8.3) name, its size in bytes, the date of its last update, and its long file name. DIR also shows the total number of bytes used by the listed files, the number of bytes left on that drive, and the total disk space.

DIR with no arguments displays information on the tables in the current directory or database. When accessing tables in the current directory, SET DBTYPE controls the files that are displayed. If SET DBTYPE is DBASE, files with .DBF extensions in the current directory are shown; if SET DBTYPE is PARADOX, .DB files are shown instead. In addition to the information normally displayed, DIR displays the number of records in each table.

The same DBF or DB tables are listed if the database chosen by SET DATABASE is a Standard table alias (one that looks at DBF and DB tables in a specific directory). If the database chosen by SET DATABASE is any other kind of alias, only the table names and the total number of tables are shown.

DIR pauses when the results pane is full and displays a dialog box prompting you to display another screenful of information.

If you have not used ON KEY or SET FUNCTION to reassign the F4 key, pressing F4 is a quick way to execute DIR.

DISKSPACE( )

Topic group Related topics

Returns the number of bytes available on the current or specified drive's disk.

Syntax

DISKSPACE([<drive expN>])

<drive expN>

A drive number from 1 to 26. For example, the numbers 1 and 2 correspond to drives A and B, respectively.

Without <drive expN> or if <drive expN> is 0, DISKSPACE( ) returns the number of bytes available on the current drive.

If <drive expN> is less than 0 or greater than 26, DISKSPACE( ) returns the number of bytes available on the drive that contains the home directory.

Description

Use DISKSPACE( ) to determine how much space is left on a disk.

DISPLAY FILES

Topic group Related topics

Displays information about files on disk in the results pane of the Command window.

Syntax

DISPLAY FILES
[[LIKE] <drive/path/filename skeleton>]
[TO FILE <filename> | ? | <filename skeleton>]
[TO PRINTER]

TO FILE <filename> | ? | *<filename skeleton>*

Directs output to a text file as well as to the results pane of the Command window. By default, *dBASE Plus* assigns a .TXT extension. The ? and <*filename skeleton*> options display a dialog box in which you specify the name of the target file and the directory to save it in.

TO PRINTER

Directs output to the printer as well as to the results pane of the Command window.

Description

DISPLAY FILES is identical to DIR, and adds the option of directing the output to a file or a printer (or both) in addition to the Command window. See DIR for details.

DISPLAY FILES is the same as LIST FILES, except that LIST FILES doesn't pause for each screenful of information but rather lists the information continuously. This makes LIST FILES more appropriate when directing output to a file or printer.

DOS

Topic group Related topics

Open an MS-DOS or Windows NT command prompt.

Syntax

DOS

Description

Use the DOS command to open an operating system command prompt. This has the same effect as choosing MS-DOS Prompt or Command Prompt from the Windows Start menu. The command prompt runs as a separate process.

To execute single operating system commands use RUN. To execute applications, use RUN( ).

eof( )

Topic group Related topics Example

Returns *true* if the file pointer is at the end of a file previously opened with *create*( ) or *open*( )

Syntax

<oRef>.*eof*( )

<oRef>

A reference to the File object that created or opened the file.

Property of

File

Description

eof( ) determines if the file pointer of the file you specify is at the end of the file (EOF), and returns *true* if it is and *false* if it is not. The file pointer is considered to be at EOF if it is positioned at the byte after the last character in the file.

You can move the file pointer to the end of the file with *seek*( ). If a file is empty, as it is when you first create a new file with *create*( ), *eof*( ) returns *true*.

ERASE

Topic group Related topics Example

Removes a file from a disk.

Syntax

ERASE <filename>

Description

ERASE is similar to the File object's *delete*( ) method, except that as a command, the file name argument is treated as a name, not a character expression. It does not require quotes unless it contains spaces or other special characters. If the name is in a variable, you must use the indirection or macro operators. Also, ERASE does not support sending a file to the Recycle Bin.

See *delete( )* for details on the operation of the command.

The DELETE FILE command is identical to ERASE.

error( )

Topic group Related topics

Returns the error number of the most recent byte-level input or output error, or 0 if the most recent byte-level method was successful.

Syntax

<oRef>.*error*( )

<oRef>

A reference to the File object that attempted the operation.

Property of

File

Description

To trap errors, call the File object method in a TRY block. Use the number that *error*( ) returns in a CATCH block to respond to errors in the byte-level methods of the File object. The following table lists the byte-level method errors that *error*( ) returns.

Error number	Cause of error
2	File or directory not found
3	Bad path name
4	No more file handles available
5	Can't access file
6	Bad file handle

8	No more directory entries available
9	Error trying to set the file pointer
13	No more disk space
14	End of file

exists( )

Topic group Related topics Example

Tests for the existence of a file. Returns *true* if the file exists and *false* if it doesn't.

Syntax

<oRef>.*exists*(<filename expC>)

<oRef>

A reference to a File object.

<filename expC>

The name of the file to search for. Wildcard characters are not allowed; you must specify the actual file name.

If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory, then in the search path(s) you specified with SET PATH, if any. If you specify a file without including its extension, *dBASE Plus* assumes no extension.

Property of

File

Description

Use *exists*( ) to determine whether a file exists. You can use either the long file name or the short file name.

FILE( )

Topic group Related topics Example

Tests for the existence of a file. Returns *true* if the file exists and *false* if it doesn't.

Syntax

FILE(<filename expC>) | FILE(":<sourceAlias>:filename expC")

Description

FILE( ) is identical to the File object's exists( ) method, except that as a built-in function, it does not require a File object to work. Also, the exists( ) method does not have the ability to evaluate a <sourceAlias>.

When you specify a <path list> using the SET PATH command, dBASE Plus searches those directories in addition to the current directory and any source alias you specify. When no SET PATH setting exists, and you don't provide the full path or source alias when you specify a file name, dBASE Plus searches for the file in the current directory only.

flush( )

Topic group Related topics

Writes to disk a file previously opened with *create*( ) or *open*( ) without closing the file. Returns *true* if successful and *false* if unsuccessful.

Syntax

<oRef>.*flush*( )

<oRef>

A reference to the File object that created or opened the file.

Property of

File

Description

Use *flush*( ) to save a file in the file buffer to disk, flush the file buffer, and keep the file open. If *flush*( ) is successful, it returns *true*.

Flushing a buffer to disk is similar to saving the file and continuing to work on it. Until you flush an open file buffer to disk, any data in the buffer is stored only in RAM (random-access memory). If the power to the computer fails or *dBASE Plus* ends abnormally, data in RAM is lost. However, if you have used *flush*( ) to write the file buffer to disk, you lose only data that was added between the time you issued *flush*( ) and the time the system failed.

To save the file to disk and close the file, use *close*( ).

FNAMEMAX( )

Topic group Related topics

Returns the maximum allowable file-name length on a given drive or volume.

Syntax

FNAMEMAX([<*expC*>])

<expC>

The drive letter (with a colon), or name of the volume, to check. If <expC> is not provided, the current drive/volume is assumed. If the drive/volume does not exist, an error occurs.

Description

FNAMEMAX( ) checks the drive or volume specified by <expC> and returns the maximum file-name length (including the dot and three-letter extension) allowed for files on that drive/volume. Typical values are:

FNAMEMAX( )	Drive type
255	Windows long file name
12	MS-DOS-compatible 8.3 name
240	Novell Netware long file name

FUNIQUE( )

Topic group Related topics Example

Creates a unique file name.

Syntax

FUNIQUE([<expC>])

<expC>

A file-name skeleton, using ? as the wildcard character (the * character is not allowed).

Description

Use FUNIQUE( ) when creating temporary files to generate a file name that is not being used by an existing file. The generated file name follows the file name skeleton you specify, with random numbers substituted for each ? character.

FUNIQUE( ) generates the new file name by replacing each wildcard character with a random number, then looking in the current or specified directory for a file name that matches the new file name. If no match is found, FUNIQUE( ) returns that name—but it does not create the file. If a match is found, FUNIQUE( ) tries again until a unique file name is found. If no combination of random numbers is successful, FUNIQUE( ) returns an empty string.

If you omit <expC>, FUNIQUE( ) returns an 8-character file name with no extension, composed entirely of random numbers, in the Windows temp directory.

GETDIRECTORY( )

Topic group Related topics

Displays a dialog box from which you can select a directory for use with subsequent commands.

Syntax

GETDIRECTORY([<directory expC>])

<directory expC>

The initial directory to appear in the dialog box. If <directory expC> is omitted, the current directory appears as the initial directory.

Description

Use GETDIRECTORY( ) to return a directory name for use in subsequent commands.

GETDIRECTORY( ) does not return a final backslash at the end of the directory name it returns. GETDIRECTORY( ) returns an empty string if the user clicks Cancel or presses Esc.

GETENV( )

Topic group Related topics

Returns the value of an operating system environment variable.

Syntax

GETENV(<expC>)

<expC>

The name of the environment variable to return.

Description

Use GETENV( ) to return the current value of an operating system environment variable.

If *dBASE Plus* can't find the environment variable specified by <expC>, it returns an emptry string.

GETFILE( )

Topic group Related topics

Displays a dialog box from which the user can choose, or enter, an existing file name, and returns this file name.

Syntax

GETFILE([<filename skeleton expC>
[, <title expC>
[, <suppress database expL>],
[<file types list expC> | <file type group name expC> (<file types list expC>)]]])

<filename skeleton expC>

A character string that matches selected file names with the wildcard characters ? and *. The GETFILE( ) dialog box initially lists only those file names in the current directory that match the file name skeleton. Without <filename skeleton expC>, the dialog box lists all file names.

<title expC>

A title displayed in the top of the dialog box. Without <title expC>, the GETFILE( ) dialog box displays the default title. If you want to specify a value for <title expC>, you must also specify a value or empty string ("") for <filename skeleton expC>.

<suppress database expL>

Whether to suppress the combobox from which you can choose a database. The default is *true*; the Database combobox is not displayed. If you want to specify a value for <suppress database expL>, you must also specify a value or empty string ("") for <filename skeleton expC> and <title expC>.

<file types list expC>

A character expression containing zero, or more, file types to be displayed in the "Files of Type" combobox. If this parameter is not specified, the following file types will be loaded into the “Files of Type” combobox:

Projects (*.prj)
Forms (*.wfm;*.wfo)
Custom Forms (*.cfm;*.cfo)
Menus (*.mnu;*.mno)
Popup (*.pop;*.poo)
Reports (*.rep;*.reo)
Custom Reports (*.crp;*.cro)
Labels (*.lab;*.lao)
Programs (*.prg;*.pro)
Tables (*.dbf;*.db)
SQL (*.sql)
Data Modules (*.dmd;*.dmo)
Custom Data Modules (*.cdm;*.cdo)
Images (*.bmp;*.ico;*.gif;*.jpg;*.jpeg;*.pje;*.xbm)
Custom Components (*.cc;*.co)
Include (*.h)
Executable (*.exe)
Sound (*.wav)
Text (*.txt)
All (*.*)

<file type group name expC>

A character expression denoting a custom or built in file type goup name to use in the dialog.

For a custom file type group name

Specify the group name followed by a list of one or more file types within parenthesis

For example, "Web Images (*.jpg,*.jpeg,*.bmp)"

For a built-in file type group name

Specify the group name followed by a set of empty parenthesis

For example, "Images ()"

dBASE will detect the left and right parenthesis and search for a matching built-in group name. If found, the list of file types associated with the built-in group will be added to the string that is added to the File Types combobox in the GetFile() or PutFile() dialog.

File Types between parenthesis must be separated by either a comma or a semi-colon and can be specified in any of the following formats:

<ext> or .<ext> or *.<ext>

where <ext> means a file extension such as jpg or txt. If a File Type List string is specified that contains unbalanced parenthesis or right parenthesis before left parenthesis, a new error will be triggered with the message:

"Unbalanced parenthesis in file type list"

Examples using file group name:

```
f = GETFILE("","Title",true,"Web Images (*.jpg,*.png,*.gif,*.bmp)")
```

or if you want to use one of the built in file type groups:

```
f = GETFILE("","Title",true,"FILE ()")
```

If an empty string is specified, "All (*.*)" will be loaded into the Files of Type combobox.

If one or more file types are specified, dBASE will check each file type specified against an internal table.

If a match is found, a descriptive character string will be loaded into the "Files of Type" combobox.

If a matching file type is not found, a descriptive string will be built, using the specified file type, in the form

`"<File Type> files (*.<File Type>)"`

and will be loaded into the "Files of Type" combobox.

When the expression contains more than one file type, they must be separated by either commas or semicolons.

File types may be specified with, or without, a leading period.

The special extension ".*" may be included in the expression to specify that "All (*.*)" be included in the Files of Type combobox.

File types will be listed in the Files of Type combobox, in the order specified in the expression.

Note: In Visual dBASE 5.x, the GETFILE( ) and PUTFILE( ) functions accepted a logical value as a parameter in the same position as the new <file types list expC> parameter. This logical value has no function in dBASE Plus. However, for backward compatibility, dBASE Plus will ignore a logical value if passed in place of the <file types list expC>

Examples of <file types list expC> syntax.can be viewed here

Description

Use GETFILE( ) to present the user with a dialog box from which they can choose an existing file or table. GETFILE( ) does not open any files.

The GETFILE( ) dialog box includes names of files whether they are currently open or closed. *dBASE Plus* returns the full path name of the file whether SET FULLPATH is ON or OFF.

By default, the dialog box opened with GETFILE( ) displays file names from the current directory the first time you issue GETFILE( ). After the first time you use and exit GETFILE( ) successfully, the subdirectory you choose becomes the default the next time you use GETFILE( ).

If <suppress database expL> is *false*, you can also choose from a list of databases. When a database is selected, the dialog box displays a list of tables in that database instead of files in the current directory.

The dialog box is a standard Windows dialog box. The user can perform many Windows Explorer-like activities in this dialog box, including renaming files, deleting files, and creating new folders. They can also right-click on a file to get its context menu. These features are disabled when the dialog is displaying tables in a database instead of files in a directory.

GETFILE( ) returns an empty string if the user chooses the Cancel button or presses Esc.

gets( )

Topic group Related topics Example

Returns a line of text from a file previously opened with *create*( ) or *open*( ).

Syntax

<oRef>.*gets*([<characters expN> [, <end-of-line expC>]])

<oRef>

A reference to the File object that created or opened the file.

<characters expN>

The number of characters to read and return before a carriage return is reached.

<end-of-line expC>

The end-of-line indicator, which can be a string of one or two characters. If omitted, the default is a hard carriage return and line feed. The following table lists standard codes used as end-of-line indicators.

Character code (decimal)	(hexadecimal)	Represents
CHR(141)	0x8D	Soft carriage return (U.S.)
CHR(255)	0xFF	Soft carriage return (Europe)
CHR(138)	0x8A	Soft linefeed (U.S.)
CHR(0)	0x00	Soft linefeed (Europe)
CHR(13)	0x0D	Hard carriage return
CHR(10)	0x0A	Hard linefeed

Use the CHR( ) function to create the <end-of-line expC> if needed. To designate the <end-of-line expC>, you must also specify the <characters expN>. If you don't want a line length limit, use an arbitrarily high number. For example:

```
cLine = f.gets( 10000, chr( 0x8d ) ) // Soft carriage return (U.S.)
```

Property of

File

Description

Use *gets*( ) to read lines from a text file. *gets*( ) reads and returns a character string from the file opened by the File object, starting at the file pointer position, and reading past but not returning the first end-of-line character(s) it encounters.

gets( ) will read characters until it encounters the end-of-line character(s) or it reads the number of characters you specify with <characters expN>, whichever comes first. If a file does not have end-of-line character(s) and you do not specify <characters expN>, *gets*( ) will read and return everything from the current file pointer position to the end of the file.

If the file pointer position is at an end-of-line character(s), *gets*( ) returns an empty string (""); the line is empty.

If *gets*( ) encounters an end-of-line character(s), it positions the file pointer at the character after the end-of-line character(s); that is, at the beginning of the next line. Otherwise, *gets*( ) positions the file pointer at the character after the last character it returns. Use *seek*( ) to move the file pointer before or after using *gets*( ).

If the file being read is not a text file, use *read*( ) instead. *read*( ) requires <characters expN> to be specified, and does not treat end-of-line characters specially.

To write a text file, use *puts*( ). *readln*( ) is identical to *gets*( ).

handle

Topic group Related topics

The operating system file handle for a file previously opened with *create*( ) or *open*( ).

Property of

File

Description

When a file is opened by the operating system, it is assigned a file handle, an arbitrary number that identifies that open file. Applications then use that file handle to refer to that file.

A File object's *handle* property reflects the file handle used by *dBASE Plus* to access a file opened with *create*( ) or *open*( ). It is a read-only property and is generally informational only. By calling methods of the File object, *dBASE Plus* internally uses the file handle to perform its operations.

HOME( )

Topic group Related topics

Returns the directory where the PLUS.exe in use is located.

Syntax

HOME( )

Description

There are two "home" directories:

The directory where *dBASE Plus* is installed, by default:

```
C:\Program Files\dBASE\Plus\
```

The directory where the actual executable file, PLUS.exe is installed. This is in the \Bin subdirectory of the installation directory, so by default, it's:

```
C:\Program Files\dBASE\Plus\Bin\
```

HOME( ) identifies the directory in which the currently running copy of PLUS.exe is located. HOME( ) returns the full path name whether SET FULLPATH is ON or OFF, and always includes the trailing backslash, as shown.

To identify the *dBASE Plus* installation home directory, use _dbwinhome.

LIST FILES

Topic group Related topics

Displays information about files on disk in the results pane of the Command window.

Syntax

LIST FILES
[[LIKE] <drive/path/filename skeleton>]
[TO FILE <filename> | ? | <filename skeleton>]
[TO PRINTER]

TO FILE <filename> | ? | *<filename skeleton>*

Directs output to a text file as well as to the results pane of the Command window. By default, *dBASE Plus* assigns a .TXT extension. The ? and *<filename skeleton>* options display a dialog box in which you specify the name of the target file and the directory to save it in.

TO PRINTER

Directs output to the printer as well as to the results pane of the Command window.

Description

LIST FILES is the same as DISPLAY FILES, except that LIST FILES doesn't pause for each screenful of information but rather lists the information continuously. This makes LIST FILES more appropriate when directing output to a file or printer.

MD

Topic group Related topics

Creates a new directory.

Syntax

MD <directory>

Description

MD is identical to MKDIR. See MKDIR for details.

MKDIR

Topic group Related topics

Creates a new directory.

Syntax

MKDIR <directory>

<directory>

The directory you want to create.

Description

Use MKDIR to create a new directory. The MD command is identical to MKDIR.

The new directory name must follow the standard naming conventions for the operating system.

If you try to make a directory that already exists or is on a path that does not exist, an error occurs.

After you create the new directory, you can use CD or SET DIRECTORY to make the new directory the current directory.

open( )

Topic group Related topics Example

Opens a specified file.

Syntax

<oRef>.*open*(<filename expC>[, <access expC>])

<oRef>

A reference to a File object.

<filename expC>

The name of the file to open. Wildcard characters are not allowed; you must specify the actual file name.

If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory, then in the search path(s) you specified with SET PATH, if any. If you specify a file without including its extension, *dBASE Plus* assumes no extension. If the named file cannot be found, an exception occurs.

<access expC>

The access level of the file being opened, as shown in the following table. The access level string is not case-sensitive, and the characters in the string may be in any order. If omitted, the default is read-only text file. Append is a more restrictive version of write; the data is always added to the end of the file

<access expC>	Access level
"R"	Read-only text file
"W"	Write-only text file
"A"	Append-only text file
"RW"	Read and write text file
"RA"	Read and append text file
"RB"	Read-only binary file
"WB"	Write-only binary file
"AB"	Append-only binary file
"RWB"	Read and write binary file
"RAB"	Read and append binary file

Property of

File

Description

Use *open*( ) to open a file with a name you specify and assign the file the level of access you specify. If *dBASE Plus* can't open the file (for example, if the file is already open), an exception occurs.

The *open*( ) method can also be used to access StdIn and StdOut, enabling direct communication with a web server. To do this, set the parameter <filename expC> to "StdIn" to receive data, or "StdOut" to transmit.

To open StdIn use: To open StdOut use:

```
fIn = new File( )     fOut = new File( )
fIn.open( "StdIn", "RA")  fOut.open( "StdOut", "RA")
```

To use other File methods, such as *read*( ) and *write*( ), first open a file with *open*( ) or *create*( ).

If you open the file with append-only or read and append access, the file pointer is positioned at the end-of-file, after the last character. For other access levels, the file pointer is positioned at the first character in the file. Use *seek*( ) to position the file pointer before reading from or writing to a file.

OS( )

Topic group Related topics

Returns the name and version number of the current operating system.

Syntax

OS( )

Description

Use OS( ) to determine the version of Windows in which *dBASE Plus* is running. OS( ) returns a character string like:

```
Windows NT version 4.00
```

with the name of the operating system, the word "version", and the version number.

Samples of what is returned by OS( ) include:

```
Windows 95  = "Windows version 4.00"
Windows 98  = "Windows version 4.10"
Windows ME  = "Windows version 4.90"
Windows NT3  = "Windows NT version 3.51"
Windows NT4  = "Windows NT version 4.00"
Windows 2000 = "Windows NT version 5.00"
Windows XP  = "Windows NT version 5.01"
Windows Vista = "Windows NT version 6.00"
Windows 7     = "Windows NT version 6.01"
```

Note

To determine which version of *dBASE Plus* is running, use VERSION( ).

path

Topic group Related topics

The full path and file name for a file previously opened with *create*( ) or *open*( ).

Property of

File

Description

When you open a file with *create*( ) or *open*( ), the path is optional. If you use *create*( ) without a path, the file is created in the current directory. If you use *open*( ) without a path, *dBASE Plus* looks for the file in the current directory, then in the search path(s) you specified with SET PATH, if any.

A File object's *path* property reflects the full path and file name for the open file. It is a read-only property.

position

Topic group Related topics

The position of the file pointer in a file previously opened with *create*( ) or *open*( ).

Property of

File

Description

A File object's *position* property reflects the current position of the file pointer. It is a read-only property. To move the file pointer, use *seek*( ). Reading and writing to a file also moves the file pointer.

The position is zero-based. The first character in the file is at position zero.

PUTFILE( )

Topic group Related topics

Displays a dialog box within which the user can choose an exisiting file to overwrite or a new file name, and returns the file name.

Syntax

PUTFILE([<title expC>
[, <filename expC>
[, <extension expC>
[, <suppress database expL>>],
[<file types list expC> | <file group name expC> (<file types list expC>)]]])

<title expC >

A title that is displayed at the top of the dialog box.

<filename expC >

The default file name that is displayed in the dialog box's entryfield. Without <filename expC >, PUTFILE( ) displays an empty entryfield.

<extension expC >

A default extension for the file name that PUTFILE( ) returns.

<suppress database expL>

Whether to suppress the combobox from which you can choose a database. The default is *true*; the Database combobox is not displayed. If you want to specify a value for <suppress database expL>, you must also specify a value or empty string ("") for <filename skeleton>, <title expC>, and <extension expC>.

<file types list expC>

A character expression containing zero, or more, file types to be displayed in the "Files of Type" combobox. If this parameter is not specified, the following file types will be loaded into the “Files of Type” combobox:

Projects (*.prj)
Forms (*.wfm;*.wfo)
Custom Forms (*.cfm;*.cfo)
Menus (*.mnu;*.mno)
Popup (*.pop;*.poo)
Reports (*.rep;*.reo)
Custom Reports (*.crp;*.cro)
Labels (*.lab;*.lao)
Programs (*.prg;*.pro)
Tables (*.dbf;*.db)
SQL (*.sql)
Data Modules (*.dmd;*.dmo)
Custom Data Modules (*.cdm;*.cdo)
Images (*.bmp;*.ico;*.gif;*.jpg;*.jpeg;*.pje;*.xbm)
Custom Components (*.cc;*.co)
Include (*.h)
Executable (*.exe)
Sound (*.wav)
Text (*.txt)
All (*.*)

If an empty string is specified, "All (*.*)" will be loaded into the Files of Type combobox.

<file type group name expC>

A character expression denoting a custom or built in file type goup name to use in the dialog.

For a custom file type group name

Specify the group name followed by a list of one or more file types within parenthesis

For example, "Web Images (*.jpg,*.jpeg,*.bmp)"

For a built-in file type group name

Specify the group name followed by a set of empty parenthesis

For example, "Images ()"

dBASE will detect the left and right parenthesis and search for a matching built-in group name. If found, the list of file types associated with the built-in group will be added to the string that is added to the File Types combobox in the GetFile() or PutFile() dialog.

File Types between parenthesis must be separated by either a comma or a semi-colon and can be specified in any of the following formats:

<ext> or .<ext> or *.<ext>

where <ext> means a file extension such as jpg or txt. If a File Type List string is specified that contains unbalanced parenthesis or right parenthesis before left parenthesis, a new error will be triggered with the message:

"Unbalanced parenthesis in file type list"

Examples using file group name:

```
f = PUTFILE("Title","","",true,"Web Images (*.jpg,*.png,*.gif,*.bmp)")
```

or if you want to use one of the built in file type groups:

```
f = PUTFILE("Title","","",true,"FILE ()")
```

If one or more file types are specified, dBASE will check each file type specified against an internal table.

If a match is found, a descriptive character string will be loaded into the "Files of Type" combobox.

If a matching file type is not found, a descriptive string will be built, using the specified file type, in the form

```
"<File Type> files (*.<File Type>)"
```

and will be loaded into the "Files of Type" combobox.

When the expression contains more than one file type, they must be separated by either commas or semicolons.

File types may be specified with, or without, a leading period.

The special extension ".*" may be included in the expression to specify that "All (*.*)" be included in the Files of Type combobox.

File types will be listed in the Files of Type combobox, in the order specified in the expression.

Note: In Visual dBASE 5.x, the GETFILE( ) and PUTFILE( ) functions accepted a logical value as a parameter in the same position as the new <file types list expC> parameter. This logical value has no function in dBASE Plus. However, for backward compatibility, dBASE Plus will ignore a logical value if passed in place of the <file types list expC>

Examples of <file types list expC> syntax.can be viewed here

Description

Use PUTFILE( ) to present the user with a dialog box from which they can choose an existing file or table or specify a new file or table name. If they choose an existing file, and SET SAFETY is ON, the user gets the standard "Replace existing file?" dialog box. If they choose "No", their choice is ignored and they are left in the PUTFILE( ) dialog box. PUTFILE( ) does not actually create or write anything to the specified file.

The PUTFILE( ) dialog box includes names of files whether they are currently open or closed. *dBASE Plus* returns the full path name of the file whether SET FULLPATH is ON or OFF.

By default, the dialog box opened with PUTFILE( ) displays file names from the current directory the first time you issue PUTFILE( ). After the first time you use PUTFILE( ) and exit successfully, the subdirectory you choose becomes the default the next time you use PUTFILE( ).

If <suppress database expL> is *false*, you can also choose from a list of databases. When a database is selected, the dialog box displays a list of tables in that database instead of files in the current directory.

The dialog box is a standard Windows dialog box. Users can perform many Windows Explorer-like activities in this dialog box, including renaming files, deleting files, and creating new folders. They can also right-click on a file to get its context menu. These features are disabled when the dialog is displaying tables in a database instead of files in a directory.

PUTFILE( ) returns an empty string if the user chooses the Cancel button or presses Esc.

puts( )

Topic group Related topics

Writes a character string, and one or two end-of-line characters, to a file previously opened with *create*( ) or *open*( ). Returns the number of characters written.

Syntax

<oRef>.*puts*(<string expC> [, <characters expN> [, <end-of-line expC>]])

<oRef>

A reference to the File object that created or opened the file.

<string expC>

The character expression to write to the specified file. If you want to write only a portion of <string expC> to the file, use the <characters expN> argument.

<characters expN>

The number of characters of the specified character expression <string expC> to write to the specified file, starting at the first character in <string expC>. If omitted, the entire string is written.

<end-of-line expC>

The end-of-line indicator, which can be a string of one or two characters. If omitted, the default is a hard carriage return and line feed. The following table lists standard codes used as end-of-line indicators.

Character code (decimal)	(hexadecimal)	Represents
CHR(141)	0x8D	Soft carriage return (U.S.)
CHR(255)	0xFF	Soft carriage return (Europe)
CHR(138)	0x8A	Soft linefeed (U.S.)
CHR(0)	0x00	Soft linefeed (Europe)
CHR(13)	0x0D	Hard carriage return
CHR(10)	0x0A	Hard linefeed

Use the CHR( ) function to create the <end-of-line expC> if needed. To designate the <end-of-line expC>, you must also specify the <characters expN>. If you don't want a line length limit, use an arbitrarily high number. For example:

```
f.puts( cLine, 10000, chr( 0x8d ) ) // Soft carriage return (U.S.)
```

Property of

File

Description

Use *puts*( ) to write text files. *puts*( ) writes a character string and one or two end-of-line characters to a file. If the file was opened in append-only or read and append mode, the string is always added to the end of the file. Otherwise, the string is written starting at the current file pointer position, overwriting any existing characters. You must have either write or append access to use *puts*( ).

puts( ) returns the number of bytes written to the file, including the end-of-line character(s). If *puts*( ) returns 0, no characters were written. Either <string expC> is an empty string, or the write was unsuccessful.

Use *error*( ) to determine if an error occurred.

When *puts*( ) finishes executing, the file pointer is located at the character immediately after the last character written, which is the end-of-line character. Successive *puts*( ) calls writes one line after another. Use *seek*( ) to move the file pointer before or after you use *puts*( ).

To write to a file that is not a text file, use *write*( ). *write*( ) does not add the end-of-line character(s). To read from a text file, use *gets*( ). *writeln*( ) is identical to *puts*( ).

read( )

Topic group Related topics Example

Returns a specified number of characters from a file previously opened with *create*( ) or *open*( ).

Syntax

<oRef>.*read*(<characters expN>)

<oRef>

A reference to the File object that created or opened the file.

<characters expN>

The number of characters to return from the specified file.

Property of

File

Description

read( ) returns the number of characters you specify from the file opened by the File object. *read*( ) starts reading characters from the current file pointer position, leaving the file pointer at the character immediately after the last character read. Use *seek*( ) to move the file pointer before or after you use *read*( ).

If the file to be read is a text file, use *gets*( ) instead. *gets*( ) looks for end-of-line characters, and returns the contents of the line, without the end-of-line character(s).

To write to a file, use *write*( ).

readln( )

Topic group Related topics

Returns a line of text from a file previously opened with *create*( ) or *open*( ).

Syntax

<oRef>.*readln*([<characters expN> [, <end-of-line expC>]])

Property of

File

Description

readln( ) is identical to *gets*( ). See *gets( )* for details.

RENAME

Topic group Related topics Example

Renames a file on disk.

Syntax

RENAME <filename> TO <new name>

Description

RENAME is identical to the File object's *rename*( ) method, except that as a command, the file name arguments are treated as names, not a character expressions. They do not require quotes unless they contain spaces or other special characters. If the name is in a variable, you must use the indirection or macro operators.

See *rename*( ) for details on the operation of the command.

rename( )

Topic group Related topics Example

Renames a file on disk.

Syntax

<oRef>.*rename*(<filename expC>, <new name expC>)

<oRef>

A reference to a File object.

<filename expC>

Identifies the file to rename (also known as the source file). <filename expC> may be a file name skeleton with wildcard characters. In that case, *dBASE Plus* displays a dialog box in which you select the file to rename.

If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory, then in the search path(s) you specified with SET PATH, if any. If you specify a file without including its extension, *dBASE Plus* assumes no extension. If the named file cannot be found, an exception occurs.

<new name expC>

Identifies the new name for the source file (also known as the target file). <new name expC> may be a file name skeleton with wildcard characters. In that case, *dBASE Plus* displays a dialog box in which you specify the name of the target file and its directory.

Property of

File

Description

rename( ) lets you change the name of a file at the operating system level.

If SET SAFETY is ON and a file exists with the same name as the target file, *dBASE Plus* displays a dialog box asking if you want to overwrite the existing file. If SET SAFETY is OFF and a file exists with the same name as the target file, an exception occurs, and the target file is not overwritten.

If you specify a different drive or directory for the target file, *dBASE Plus* moves the source file to that location. When a path is not specified, the target file is moved to the current directory.

rename( ) does not automatically rename the auxiliary files associated with table files, such as indexes and memo files. For example, it does not rename the MDX or DBT files associated with a DBF file. When renaming tables, use the Database object's *renameTable*( ) method.

RMDIR

Topic group Related topics

Removes a directory.

Syntax:

RMDIR <directory>
or
RD <directory>

<directory>
- The directory you want to remove. This can be a full directory path or a relative directory path.

Description

RMDIR attempts to remove the specified directory. It can only remove a directory if the directory is empty and is not currently in use by any program.

RD works exactly the same way as RMDIR.

For Example:

RMDIR mydir
- Will attempt to remove directory mydir

RMDIR C:\somefolder\otherfolder
- Will attempt to remove directory otherfolder

RD mydir
- Will attempt to remove directory mydir

RD C:\somefolder\otherfolder
- Will attempt to remove directory otherfolder

RUN

Topic group Related topics Example

Executes a program or operating system command from within *dBASE Plus*.

Syntax

RUN <command>

<command>

A command recognized by your operating system.

Description

Use RUN to execute a single operating system command or program from within *dBASE Plus*. Enter commands and file names exactly as you would when working in the command prompt; do not enclose them in quotes. ! is equivalent to RUN.

RUN opens a command prompt in the current directory and executes <command>. The command prompt automatically closes when the program or command is finished. Commands

and programs launched by RUN execute as a separate task, as if you had started that task from the Start menu. *dBASE Plus* continues to run on its own.

To open a command prompt so you can enter multiple commands yourself, use the DOS command. To execute a Windows application, use RUN( ) instead; it does not open a command prompt window.

RUN( )

Topic group Related topics

Executes a program or operating system command from within *dBASE Plus*, returning the instance handle of the program.

Syntax

RUN([<direct expL>,] <command expC>)

<direct expL>

Determines whether RUN( ) runs a Windows program directly (*true*) or through a command prompt (*false*). If <command expC> is not a Windows program, <direct expL> must be *false*, or RUN( ) has no effect. If you omit <direct expL>, *dBASE Plus* assumes a value of *false*.

<command expC>

A Windows program name or a command recognized by your operating system.

Description

Use RUN( ) to execute another Windows program or an operating system command from within *dBASE Plus*.

To run another Windows program, <direct expL> should be *true*; otherwise, a separate command prompt is opened first, and you cannot get the returned instance handle.

seek( )

Topic group Related topics Example

Moves the file pointer in a file previously opened with *create*( ) or *open*( ), and returns the new position of the file pointer.

Syntax

<oRef>.*seek*(<offset expN> [, <position expN>])

<oRef>

A reference to the File object that created or opened the file.

<offset expN>

The number of bytes to move the file pointer in the specified file. If <offset expN> is negative, the file pointer moves toward the beginning of the file. If <offset expN> is 0, the file pointer moves to the position you specify with <position expN>. If <offset expN> is positive, the file pointer moves toward the end of the file or beyond the end of the file.

<position expN>

The number 0, 1, or 2, indicating a position relative to the beginning of the file (0), to the file pointer's current position (1), or to the end of the file (2). The default is 0.

Property of

File

Description

seek( ) moves the file pointer in the file you specify relative to the position specified by <position_expN>, and returns the resulting position of the file pointer as an offset from the beginning of the file. The File object's *position* property is also updated with this new position. If an error occurs, seek( ) returns –1.

The movement of the file pointer is relative to the beginning of the file unless you specify otherwise with <position expN>. For example, *seek*(5) moves the file pointer five characters from the beginning of the file (the 6th character) while *seek*(5,1) moves it five characters forward from its current position. You can move the file pointer beyond the end of the file, but you can't move it before the beginning of the file.

To move the file pointer to the beginning of a file, use *seek*(0). To move it to the end of a file, use *seek*(0, 2). To move to the last character in a file, use *seek*(–1,2).

gets( ), *puts*( ), *read*( ), and *write*( ) also move the file pointer as they read from or write to the file.

SET DIRECTORY

Topic group Related topics

Changes the current drive or directory.

Syntax

SET DIRECTORY TO [<path>]

<path>

The new drive and/or path. Quotes (single or double) are required if the path contains spaces or other special characters; optional otherwise.

Description

SET DIRECTORY works like CD, except SET DIRECTORY TO (with no argument) returns you to the HOME( ) directory, while CD with no argument displays the current directory.

To get the current directory, use SET("DIRECTORY").

SET FULLPATH

Topic group Related topics

Specifies whether functions that return file names return the full path with the file name.

Syntax

SET FULLPATH on | OFF

Description

Use SET FULLPATH ON when you need to have functions or methods such as *shortName*( ), return a file name with its full path. When SET FULLPATH is OFF, these functions include the drive letter (and colon) with the file name only.

Some functions, such as GETFILE( ), always return the full path, regardless of SET FULLPATH.

SET PATH

Topic group Related topics

Specifies the directory search route that *dBASE Plus* follows to find files that are not in the current directory.

Syntax

SET PATH TO [<path list>]

<path list>

A list of (optional) drives and directories indicating the search path—one or more drives and directories you want *dBASE Plus* to search for files. Separate each directory path name with commas, semicolons, or spaces. If the path name contains spaces or other special characters, the path name should be enclosed in quotes.

Description

Use SET PATH to establish a search path to access files located on directories other than the current directory. When no SET PATH setting exists and you don't provide the full path name when you specify a file name, *dBASE Plus* searches for that file only in the current directory.

The order in which you list drives and directories with SET PATH TO <path list> is the order *dBASE Plus* searches for a file in that search path. Use SET PATH when an application's files are in several directories.

SET PATH TO without the option <path list> resets the search path to the default value (no path).

shortName( )

Topic group Related topics

Returns the short (8.3) name of a file.

Syntax

<oRef>.*shortName*(<filename expC>)

<oRef>

A reference to a File object.

<filename expC>

The name of the file to check. Wildcard characters are not allowed; you must specify the actual file name.

If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory, then in the search path(s) you specified with SET PATH, if any. If you specify a file without including its extension, *dBASE Plus* assumes no extension. If the named file cannot be found, an exception occurs.

Property of

File

Description

shortName( ) checks the file specified by <flename expC> and returns a name for the file following the DOS file-naming convention (eight-character file name, three-character extension). If SET FULLPATH is ON, the path is also returned.

size( )

Topic group Related topics Example

Returns the size of a file in bytes.

Syntax

<oRef>.*size*(<filename expC>)

<oRef>

A reference to a File object.

<filename expC>

The name of the file to check. Wildcard characters are not allowed; you must specify the actual file name.

If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory, then in the search path(s) you specified with SET PATH, if any. If you specify a file without including its extension, *dBASE Plus* assumes no extension. If the named file cannot be found, an exception occurs.

Property of

File

Description

Use *size*( ) to determine the size of a file on disk.

With the byte-level access methods of the File object, *dBASE Plus* doesn't update the size on the file recorded on the disk until you *close*( ) the file.

time( )

Topic group Related topics Example

Returns the time stamp for a file, the time the file was last modified.

Syntax

<oRef>.*time*(<filename expC>)

<oRef>

A reference to a File object.

<filename expC>

The name of the file to check. Wildcard characters are not allowed; you must specify the actual file name.

If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory, then in the search path(s) you specified with SET PATH, if any. If you specify a file without including its extension, *dBASE Plus* assumes no extension. If the named file cannot be found, an exception occurs.

Property of

File

Description

Use *time*( ) to determine the time of day when the last change was made to a file on disk. *time*( ) returns the time as a character string.

When you update a file, *dBASE Plus* changes the file's time stamp to the current operating system time when the file is written to disk. For example, when the user edits a DB table, *dBASE Plus* changes the time stamp on the table file when the file is closed. *time*( ) reads the time stamp and returns its current value.

To get the time the file was created, use *createTime*( ). For the date the file was last modified, use *date*( ).

TYPE

Topic group Related topics

Display the contents of a text file.

Syntax

TYPE <filename 1> | ? | <filename skeleton 1>
[MORE]
[NUMBER]
[TO FILE <filename 2> | ? | <filename skeleton 2>] | [TO PRINTER]

<filename> | ? | <filename skeleton>

The file whose contents to display. TYPE ? and TYPE <filename skeleton> display a dialog box from which you can select a file. If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory, then in the path you specify with SET PATH. You must specify a file-name extension.

MORE

Pauses output when it fills the Command window; otherwise, the output scrolls through the Command window to the end of the file.

NUMBER

Precedes each line of output with its line number.

TO FILE <filename 2> | ? | <filename skeleton>

Directs output to the text file <filename 2>, as well as to the results pane of the Command window. By default, *dBASE Plus* assigns a .TXT extension to <filename 2> and saves the file in the current directory. The ? and <filename skeleton> options display a dialog box in which you specify the name of the target file and the directory to save it in.

TO PRINTER

Directs output to the printer, as well as to the results pane of the Command window.

Description

Use TYPE to display the contents of text files. All program files in *dBASE Plus* are text files that you can display with TYPE.

If you TYPE a file TO FILE or TO PRINTER, *dBASE Plus* adds two lines of output at the beginning of the saved or printed output if SET HEADINGS is ON. The first line is a blank line, and the second line contains the full path name and date stamp of the source file. If you specify

NUMBER, these two lines are not numbered; numbering begins with 1 at the first actual line of the source file.

If you specify MORE and cancel output before completion, *** INTERRUPTED *** appears in the results pane of the Command window, but does not appear in the incomplete saved or printed output.

If SET SAFETY is ON and a file exists with the same name as the target file, *dBASE Plus* displays a dialog box asking if you want to overwrite the file. If SET SAFETY is OFF, any existing file with the same name is overwritten without warning.

VALIDDRIVE( )

Topic group Related topics Example

Returns *true* if the specified drive exists and can be read. Returns *false* if the specified drive does not exist or cannot be read.

Syntax

VALIDDRIVE(<drive expC>)

<drive expC>

The drive to be tested, which can be either:

A drive letter, optionally followed by a colon, or
The UNC name for a drive

Description

Use VALIDDRIVE( ) to determine if a specified drive exists and is ready before using CD, SET DEFAULT, SET DIRECTORY or SET PATH. VALIDDRIVE( ) is also useful if your program copies files to or from a drive, or includes drive letters in any file names.

VALIDDRIVE( ) can verify any drive specified, including drives created by partitioning a hard disk and mapped network drives. Checking for a floppy disk or network drive takes a few seconds, so you should display a message before you check.

write( )

Topic group Related topics Example

Writes a character string to a file previously opened with *create*( ) or *open*( ). Returns the number of characters written.

Syntax

<oRef>.*write*(<expC> [, <characters expN>])

<oRef>

A reference to the File object that created or opened the file.

<expC>

The character expression to write to the specified file. If you want to write only a portion of <string expC> to the file, use the <characters expN> argument.

<characters expN>

The number of characters of the specified character expression <string expC> to write to the specified file, starting at the first character in <string expC>. If omitted, the entire string is written.

Property of

File

Description

write( ) writes a character string to a file. If the file was opened in append-only or read and append mode, the string is always added to the end of the file. Otherwise, the string is written starting at the current file pointer position, overwriting any existing characters. You must have either write or append access to use *write*( ).

write( ) returns the number of bytes written to the file. If *write*( ) returns 0, no characters were written. Either <expC> is an empty string, or the write was unsuccessful.

Use *error*( ) to determine if an error occurred.

When *write*( ) finishes executing, the file pointer is located at the character immediately after the last character written. Use *seek*( ) to move the file pointer before or after you use *write*( ).

To write to a text file, use *puts*( ). *puts*( ) automatically adds the end-of-line character(s).

To read from a file, use *read*( ).

writeln( )

Topic group Related topics

Writes a character string and one or two end-of-line characters to a file previously opened with *create*( ) or *open*( ). Returns the number of characters written.

Syntax

<oRef>.*writeln*(<string expC> [, <characters expN> [, <end-of-line expC>]])

Property of

File

Description

writeln( ) is identical to *puts*( ). See *puts( )* for details.

_dbwinhome

Topic group Related topics Example

Contains the home directory of the currently running instance of *dBASE Plus*.

Description

There are two "home" directories:

The directory where *dBASE Plus* is installed, by default:

```
C:\Program Files\dBASE\Plus\
```

The directory where the actual executable file, PLUS.exe is installed. This is in the \Bin subdirectory of the installation directory, so by default, it's:

```
C:\Program Files\dBASE\Plus\Bin\
```

_dbwinhome contains the installation home directory, from which you can access all subdirectories. _dbwinhome contains the full path name whether SET FULLPATH is ON or OFF, and always includes the trailing backslash, as shown.

_dbwinhome is read-only.

To identify where the currently running instance of PLUS.exe is located, use HOME( ).

Xbase

Xbase introduction

Topic group

Every Xbase command and function includes a section that lists the OODML (object-oriented data manipulation language) equivalent, when one exists.

The examples in this section are mostly data processing and utility code. Data entry in *dBASE Plus* is done at another level, either using the form-based events that are melded into the Xbase worksets, or the *dBASE Plus* data objects, which replace most Xbase functionality and provide powerful new object-oriented capabilities.

The examples also do not use any new dBL syntax, and thus are compatible with older versions of dBASE.

Common command elements

Topic group

The following sections detail command elements that are common to many Xbase commands and functions.

Filenames
Aliases
Command scope

Filenames

Topic group

Filenames are required for many Xbase commands. The filename may refer to a file on disk or a table in a database. A filename is indicated by <filename> in the syntax diagram and may be any one of the following forms:

- A filename, without the extension. When the filename refers to a table, *dBASE Plus* will assume the extension specified by the SET DBTYPE command (.DBF for dBASE and .DB for Paradox), which can be overridden in most commands with the command's TYPE clause. If the SET DATABASE command has been used to set a server database as the default, then the table name will be used as-is, without an extension. When the filename is not a table, there is always a default extension, which is listed in each command description.
- A filename, with the extension.
- A table in a database. Use the BDE Administrator to create database aliases. Specify the database alias in colons before the table name as follows:

  ```
  :databaseAlias:tableName
  ```

  If the database is not already open, *dBASE Plus* displays a dialog box in which you specify the parameters, such as a login name and password, necessary to establish a connection to that database.
- A filename skeleton. Use the ? and * as wildcard characters. A single ? is the same as *, meaning any filename. A dialog box is displayed from which you can choose a table, either a file on disk or a table from a database.

In all cases, the <filename> may be enclosed in string delimiters (single quotes, double quotes, or square brackets). Delimiters are required if <table name> contains spaces or other special characters. If the <filename> is contained in a variable and is not defined as an expression—functions expect filenames that are character expressions, commands do not—use the parentheses as indirection operators on the variable containing the <filename>.

If the <filename> refers to a file and does not contain a path and the file is not found in the current directory, then the path specified by SET PATH is also searched.

In many commands, the <filename> does not have to be specified in the statement. If it is omitted, *dBASE Plus* will display a dialog box from which you can choose a file to execute the command.

For commands that specifically create files and not tables, the database options are not allowed. If a dialog box is displayed, it will not include the controls to choose a database.

If you are about to overwrite a file, you will get a confirmation dialog box if SET SAFETY is ON. If SET SAFETY is OFF, the file will be overwritten without a warning.

Aliases

Topic group

While some commands work only in the current work area, others allow you to specify the work area in which they perform their function. Work areas are referenced by their alias, which may take one of the following forms:

The work area number, from 1 to 225

A character string that contains a single letter from A through J, which correspond to work area 1 through 10. This is supported for compatibility.

A character string containing the name of the work area: the name of the table, or the alias assigned to the work area when the table was opened. See the USE command for information on assigning aliases.

When using a letter or work area name as the alias in a function, the alias must be a character expression, usually the string enclosed in string delimiters. In a command, the delimiters are optional and usually not used, unless the alias contains spaces or other special characters. In addition to the normal string delimiters (single quotes, double quotes, and square brackets), colons may be used to delimit aliases in commands.

The alias option is indicated by <alias> in the syntax tables. When you do not specify an alias, the command or function works on the current work area.

Command scope

Topic group

Many Xbase commands have a scope option (not to be confused with the scope resolution operator) that dictates which records to process. The scope honors the current index order, filter, and key constraints. Three clauses comprise a command's scope:

<scope>

FOR <for condition>

WHILE <while condition>

There are four different options for <scope>:

ALL

All records, starting with the first.

REST

Starting with the current record, processes all subsequent records in the table

NEXT <expN>

Starting with the current record, processes the next <expN> records. NEXT 1 processes the current record only.

RECORD <bookmark>

The individual record referenced by the bookmark <bookmark>. You may also specify a record number for DBF tables.

Different commands have different default scopes. In conjunction with <scope>, many commands have one or both of the following conditional clauses:

FOR <for condition>

Specifies a condition that must evaluate to *true* for each record to be processed. If the <for condition> fails, that record is skipped and the next record is tested.

WHILE <while condition>

Specifies a condition that must evaluate to *true* for processing to continue. The test is performed before processing each record. If the <while condition> fails, processing stops.

If you specify a FOR clause, the default scope of the command becomes ALL. If you specify a WHILE clause, with or without a FOR clause, the default scope of the command becomes REST.

ALIAS( )

Topic group Related topics Example

Returns the alias name of the current or a specified work area.

Syntax

ALIAS([<alias>])

<alias>

The work area you want to check. (If <alias> is a work area alias name, there is no reason to use this function because that alias name is what the function will return.)

Description

ALIAS( ) returns the alias name of any work area within the current workset, in all uppercase. If no table is opened in the specified work area, ALIAS( ) returns an empty string ("").

Routines that do work in other work areas usually save the current work area before switching, and then switch back when done. Use ALIAS( ) to get the name of the current work area, then switch back using the SELECT command.

OODML

There is no concept of the "current" Query object. You may refer to any Query object at any time through its object reference.

APPEND

Topic group Related topics Example

Adds a new record to a table.

Syntax

APPEND [BLANK]

BLANK

Adds a blank record to the end of the table and makes the blank record the current record.

Description

APPEND displays the currently selected table in an auto-generated data entry form and puts the form in Append mode. This has the same effect as using the EDIT command to display the data entry form and manually choosing Add Row from the menu or toolbar. This interactive APPEND is rarely used in applications because you have no control over the appearance of the data entry form.

The APPEND BLANK command adds a blank record to the current table and positions the record pointer on the new record, but it doesn't display a window to edit the data. This is often done in an older style of dBASE programming, and is typically followed by REPLACE statements to store values into the newly-created record.

When accessing SQL tables, some database servers do not allow you to enter blank records. Also, constraints on tables created with non-null fields, including DBF7 tables, prevent entering records with fields left blank. In these cases, APPEND BLANK will fail and cause an error.

OODML

Use the Rowset object's *beginAppend( )* method. While APPEND BLANK creates a blank record first that you must delete if you decide to discard the new record, *beginAppend*( ) blanks the row buffer and creates a new row only if the row is modified and saved.

APPEND AUTOMEM

Topic group Related topics Example

Adds a new record to a table using the values stored in automem variables.

Syntax

APPEND AUTOMEM

Description

APPEND AUTOMEM adds a new record to the currently selected table and then replaces the value of fields in the table with the contents of corresponding automem variables. Automem variables are variables that have the same names and data types as the fields in the current table. Automem variables must be private or public; they cannot be local or static. If a field does not have a matching variable, that field is left blank.

APPEND AUTOMEM is used as part of data entry in an older style of dBL programming. In *dBASE Plus*, controls in data entry forms are *dataLink*ed to fields; there is no need for a set of corresponding variables. APPEND AUTOMEM is also used for programmatically adding records to a table. It is more convenient than using APPEND BLANK and REPLACE.

To use APPEND AUTOMEM to add records to a table, first create a set of automem variables. The USE...AUTOMEM command opens a table and creates the corresponding empty automem variables for that table. CLEAR AUTOMEM creates a set of empty automem variables for the current table or reinitializes existing automem variables to empty values. STORE AUTOMEM

copies the values in the current record to automem variables. You may also create the individual variables manually.

When referring to the value of automem variables you need to prefix the name of an automem variable with M-> to distinguish the variable from the corresponding fields, which have the same name. The M-> prefix is not needed during variable assignment; the STORE command and the = and := operators do not work on Xbase fields.

Note: Read-only field type - Autoincrement

Because APPEND AUTOMEM and REPLACE AUTOMEM write values to your table, the contents of the read-only field type, Autoincrement, must be released before using either of these commands. In the following example, the autoincrement field is represented by "myAutoInc":

```
use table1 in 1
use table2 in 2
select 1     // navigate to record
store automem
release m->myAutoInc
select 2
append automem
```

OODML

The Rowset object contains an array of Field objects accessed through its *fields* property. These Field objects have *value* properties that may be programmed like variables.

APPEND FROM

Topic group Related topics

Copies records from an existing table to the end of the current table.

Syntax

APPEND FROM <filename>
[FOR <condition>]
[[TYPE] DBASE | PARADOX | SDF |
 DELIMITED [WITH <char> | BLANK]]
[REINDEX]

<filename>

The name of the file whose records you want to append to the current table.

FOR <condition>

Restricts APPEND FROM to records in <filename> that meet <condition>. You can specify a FOR <condition> only for fields that exist in the current table. *dBASE Plus* pretends that the record is appended, then evaluates the <condition>. If it fails, the record is not actually appended.

**[TYPE] DBASE | PARADOX | SDF |
DELIMITED [WITH <char> | BLANK]**

Specifies the default file extension, and for text files, the text file format. For example, if you specify a .DBF file as the <filename> and TYPE PARADOX, the TYPE is ignored because the file is really a dBASE file. The TYPE keyword is included for readability only; it has no effect on the operation of the command. The following table provides a description of the different file formats that are supported:

Type	Description

DBASE	A dBASE table. If you don't include an extension for <filename>, *dBASE Plus* assumes a .DBF extension.
PARADOX	A Paradox table. If you don't include an extension for <filename>, *dBASE Plus* assumes a .DB extension.
SDF	A System Data Format text file. Records in an SDF file are fixed-length, and the end of a record is marked with a carriage return and a linefeed. If you don't specify an extension, *dBASE Plus* assumes .TXT.
DELIMITED	A text file with fields separated by commas. These files are also referred to as CSV (Comma Separated Value) files. Character fields may be delimited with double quotation marks; the quotes are required if the field itself contains a comma. Each carriage return and linefeed indicates a new record. If you don't specify an extension, *dBASE Plus* assumes .TXT.
DELIMITED WITH <char>	Indicates that character data is delimited with the character <char> instead of with double quotes. For example, if delimited with a single quote instead of a double quote, the clause would be: DELIMITED WITH '
DELIMITED WITH BLANK	Indicates that data is separated with spaces instead of commas, with no delimiters.

REINDEX

Rebuilds all open index files after APPEND FROM finishes executing. Without REINDEX, *dBASE Plus* updates all open indexes after appending each record from <filename>. When the current table has multiple open indexes or contains many records, APPEND FROM executes faster with the REINDEX option.

Description

Use the APPEND FROM command to add data from another file or table to the end of the current table. You can append data from dBASE tables or files in other formats. Data is appended to the current table in the order in which it is stored in the file you specify.

When you specify a table as the source of data, fields are copied by name. If a field in the current table does not have a matching field in the source table, those fields will be blank in the appended records. If the field types do not match, type conversion is attempted. For example, if a field named ID in the current table is character field, but the ID field in the source table is numeric, the number will be converted into a string when it is appended.

When appending text files, SDF or DELIMITED, there is no data type in the source file; everything is a string. For non-character fields, the strings should be in the following format to match the data type in the table:

For logical or boolean fields, the letters T, t, Y, and Y indicate *true*. All other letters and blanks are considered *false*.
Dates must be in the format YYYYMMDD.

If the field of the current table is shorter than the matching field of the source table, *dBASE Plus* truncates the data.

If SET DELETED is OFF, *dBASE Plus* adds records from a source dBASE table that are marked for deletion and doesn't mark them for deletion in the current table. If SET DELETED is ON, *dBASE Plus* doesn't add records from a source dBASE table that are marked for deletion.

When importing data from other files, remove column headings and leading blank rows and columns; otherwise, this data is also appended.

OODML

Use the UpdateSet object's *append( )* or *appendUpdate( )* method to append data from other tables.

APPEND FROM ARRAY

Topic group Related topics

Adds to the current table one or more records containing data stored in a specified array.

Syntax

APPEND FROM ARRAY <array>
[FIELDS <field list>]
[FOR <condition>]
[REINDEX]

<array>

A reference to the array containing the data to store in the current table as records.

FIELDS <field list>

Appends <array> data only to the fields in <field list>. Without FIELDS <field list>, APPEND FROM ARRAY appends to all the fields in the table, starting with the first field.

FOR <condition>

Restricts APPEND FROM ARRAY to array rows in <array> that meet <condition>. The FOR <condition> should reference the fields in the current table. *dBASE Plus* pretends that the record is appended, then evaluates the <condition>. If it fails, the record is not actually appended.

REINDEX

Rebuilds open indexes after all records have been changed. Without REINDEX, *dBASE Plus* updates all open indexes after appending each record from <array>. When the current table has multiple open indexes or contains many records, APPEND FROM ARRAY executes faster with the REINDEX option.

Description

APPEND FROM ARRAY treats one- and two-dimensional arrays as tables, with columns corresponding to fields and rows corresponding to records. A one-dimensional array works as a table with only one row; therefore, you can append only one record from a one-dimensional array. A two-dimensional array works as a table with multiple rows; therefore, you can append as many records from a two-dimensional array as it has rows.

When you append data from an array to the current table, *dBASE Plus* appends each array row as a single record. If the table has more fields than the array has columns, the excess fields are left empty. If the array has more columns than the table has fields, the excess columns are ignored. The data in the first column is added to the first field's contents, the data in the second column to the second field's contents, and so on.

The data types of the array must match those of corresponding fields in the table you are appending. If the data type of an array element and a corresponding field don't match, *dBASE Plus* returns an error.

If the current table has a memo field, *dBASE Plus* ignores this field. For example, if the second field is a memo field, *dBASE Plus* adds the data in the array's first column to the first field's contents, and the data in the array's second column to the third field's contents.

Use APPEND FROM ARRAY as an alternative to automem variables when you need to transfer data between tables where the structures are similar but the field names are different.

OODML

Use two nested loops to first call the Rowset object's *beginAppend( )* method to create the new rows, and then to copy the elements of the array into the *value* properties of the Field objects in the rowset's *fields* array.

APPEND MEMO

Topic group Related topics Example

Appends a text file to a memo field.

Syntax

APPEND MEMO <memo field> FROM <filename>
[OVERWRITE]

<memo field>

The memo field to append to.

FROM <filename>

The text file to append. The default extension is .TXT.

OVERWRITE

Erases the contents of the current record memo field before copying the contents of <filename>.

Description

Use the APPEND MEMO command to insert the contents of a text file into a memo field. You may use an alias name and the alias operator (that is, alias->memofield) to specify a memo field in the current record of any open table.

APPEND MEMO is identical to REPLACE MEMO, except that APPEND MEMO defaults to appending the file to the current contents of the memo field and has the option of overwriting, while REPLACE MEMO is the opposite.

While memo fields may contain types of information other than text, binary fields are recommended for storing images, sound, and other user-defined binary type information. Use OLE fields for linking to OLE documents from other Windows applications.

OODML

Use the Field object's *replaceFromFile( )* method.

AVERAGE

Topic group Related topics Example

Computes the arithmetic mean (average) of specified numeric fields in the current table.

Syntax

AVERAGE [<exp list>]
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]
[TO <memvar list> | TO ARRAY <array>]

<exp list>

The numeric fields, or expressions involving numeric fields, to average.

<scope>
FOR <condition 1>
WHILE <condition 2>

The scope of the command. The default scope is ALL.

TO <memvar list> | TO ARRAY <array>

Initializes and stores averages to the variables (or properties) of <memvar list> or stores averages to the existing array <array>. If you specify an array, each field average is stored to elements in the order in which you specify the fields in <exp list>. If you don't specify <exp list>, each field average is stored in field order. <array> can be a single- or multidimensional array; the array elements are accessed via their element numbers, not their subscripts.

Description

The AVERAGE command computes the arithmetic means (averages) of numeric expressions and stores the results in specified variables or array elements. If you store the values in variables, the number of variables must be exactly the same as the number of fields or expressions averaged. If you store the values in an array, the array must already exist, and the array must contain at least as many elements as the number of averaged expressions.

If SET TALK is ON, AVERAGE also displays its results in the results pane of the Command window. The SET DECIMALS setting determines the number of decimal places that AVERAGE displays. Numeric fields in blank records are evaluated as zero. To exclude blank records, use the ISBLANK( ) function in defining a FOR condition. EMPTY( ) excludes records in which a specified expression is either 0 or blank.

OODML

Loop through the rowset to calculate the average.

BEGINTRANS( )

Topic group Related topics

Begins transaction logging.

Syntax

BEGINTRANS([<database name expC> [,<isolation level expN>]])

<database name expC>

The BDE alias of the SQL database in which to begin the transaction.

If <database name expC> is omitted but a SET DATABASE statement has been issued, BEGINTRANS( ) refers to the database in the SET DATABASE statement.

If <database name expC> is omitted and no SET DATABASE statement has been issued, the default database, which supports DBF and DB tables is used.

<isolation level expN>

Specifies a pre-defined server-level transaction isolation scheme.

Valid values for <isolation level> are:

Value	Description
0	Server's default isolation level
1	Uncommitted changes read (dirty read)
2	Committed changes read (read committed)
3	Full read repeatability (repeatable read)

<isolation level> is not supported for DBF and DB tables.

If an invalid value is given for <isolation level>, a "Value out of range" error is generated.

The <isolation level> is server-specific; a "Not supported" error will result from the database engine if an unsupported level is specified.

Note

If you include <database name expC> when you issue BEGINTRANS( ), you must also include it in subsequent COMMIT( ) or ROLLBACK( ) statements within that transaction. If you don't, *dBASE Plus*. ignores the COMMIT( ) or ROLLBACK( ) statement.

Description

Separate changes that must be applied together are considered to be a transaction. For example, transferring money from one account to another means debiting one account and crediting another. If for whatever reason one of those two changes cannot be done, the whole transaction is considered a failure and any change that was made must be undone.

Transaction logging records all the changes made to all the tables in a database. If no errors are encountered while making the individual changes in the transaction, the transaction log is cleared with COMMIT( ) and the transaction is done. If an error is encountered, all changes made so far are undone by calling ROLLBACK( ).

All locks made during a transaction are maintained until the transaction is completed. This ensures that no one else can make any changes until the transaction is committed or abandoned.

You can't nest transactions with BEGINTRANS( ). If you issue BEGINTRANS( ) against an SQL database that does not support transactions, or if a server error occurs, BEGINTRANS( ) returns *false*. Otherwise, it returns *true*. If BEGINTRANS( ) returns *false*, use SQLERROR( ) or SQLMESSAGE( ) to determine the nature of the server error that might have occurred.

OODML

Call the *beginTrans( )* method of the Database object.

BINTYPE( )

Topic group Related topics

Returns the predefined type number of a specified binary field.

Syntax

BINTYPE([<field name>])

<field name>

The name of a field in the current table.

Description

BINTYPE( ) returns the predefined binary type number of a binary field in the current table. Using this command, you can determine the type of data stored in the field. The values returned by BINTYPE( ) are the following:

Predefined binary type numbers	Description
1 to 32K – 1 (1 to 32,767)	User-defined file types
32K (32,768)	.WAV files
32K + 1 (32,769)	.BMP and .PCX files

BINTYPE( ) returns an error if a non-binary field is specified. It returns a value of 0 if the binary field is empty.

OODML

No direct equivalent. You may be able to ascertain the data type by examining the data in the *value* of the Field object.

BLANK

Topic group Related topics Example

Fills fields in records with blanks.

Syntax

```
BLANK
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]
[FIELDS
 <field list> | [LIKE <skeleton 1>] [EXCEPT <skeleton 2>]]
[REINDEX]
```

<scope>
FOR <condition 1>
WHILE <condition 2>

The scope of the command. The default scope is NEXT 1, the current record only.

FIELDS <field list> | LIKE <skeleton 1> | EXCEPT <skeleton 2>

The fields to blank. Without FIELDS, BLANK replaces all fields. If you specify FIELDS LIKE <skeleton 1>, the BLANK command restricts the fields that are blanked to the fields that match <skeleton 1>. Conversely, if you specify FIELDS EXCEPT <skeleton 2>, the BLANK command makes all fields blank except those whose names match <skeleton 2>.

REINDEX

Rebuilds all open indexes after BLANK finishes executing. Without REINDEX, *dBASE Plus* updates all open indexes after each record is made blank. When the current table has multiple open indexes or contains many records, BLANK executes faster with the REINDEX option.

Description

Use BLANK to blank-out fields or records in the current table. BLANK has the same effect as using REPLACE on each field with a *null* value. For DBF7, DB, and SQL tables, the fields are replaced with *null* values. For earlier versions of DBF tables, the fields are replaced with blanks (spaces). EMPTY( ) and ISBLANK( ) return *true* for a field whose value has been replaced using BLANK. BLANK fills an existing record with the same values as APPEND BLANK. Updates to open indexes are performed after each record or a set of records is blanked.

The distinction between blank and zero values in numeric fields can be significant when you use commands such as AVERAGE and CALCULATE.

For earlier DBF tables, blank numeric fields evaluate to zero and blank logical or boolean fields evaluate to *false*. In DBF7 tables, which support true null values, the value of the field is *null*, although some commands may display the null value as zero or *false*.

OODML

Use a loop to assign *null* values to the *value* properties of the Field objects.

BOF( )

Topic group Related topics

Indicates if the record pointer in a table is at the beginning of the file.

Syntax

BOF([<alias>])

<alias>

The work area you want to check.

Description

BOF( ) returns *true* when the record pointer has just moved before the first logical record of the table in the specified work area; otherwise, it returns *false*. For example, if you issue SKIP -1 when the record pointer is on the first record, BOF( ) returns *true*. If you attempt to navigate backwards when BOF( ) is *true*, an error occurs.

However, unlike EOF( ), the record pointer can never stay before the first record. After the record pointer has moved past the first record, it is automatically moved back to the first record, even though BOF( ) remains *true*. Subsequent navigation will cause BOF( ) to return *false* unless the navigation moves the record pointer before the first record again.

When you first USE a table, BOF( ) can never be *true*, but EOF( ) can if the table is empty, or you are using a conditional index with no matching records.

If no table is open in the specified work area, BOF( ) also returns *false*.

OODML

The Rowset object's *endOfSet* property is *true* when the row pointer is past either end of the rowset. Unlike BOF( ) and EOF( ), there is symmetry with the *endOfSet* property. You can determine which end you're on based on the direction of the last navigation.

There is also an *atFirst( )* method that determines whether you are on the first row in the rowset.

BOOKMARK( )

Topic group Related topics

Returns a bookmark for the current record.

Syntax

BOOKMARK([<alias>])

<alias>

The work area you want to check.

Description

BOOKMARK( ) returns a value for the current record. The value returned by BOOKMARK( ) is of a special unprintable data type called bookmark. BOOKMARK( ) returns an empty bookmark if no table is open in the current work area.

When used with the GO command, bookmarks let you navigate to particular records.

Unlike record numbers, which work only with DBF tables, bookmarks work with all tables, including DBFs. Bookmarks are only guaranteed to be valid for the table from which they are created.

Bookmark values can be used in all commands and functions that can otherwise use a record number, and with relational operators to check for equality and relative position in a table.

OODML

Use the Rowset object's *bookmark( )* method.

BROWSE

Topic group Related topics

Provides display and editing of records in a table format.

Syntax

BROWSE
[COLOR <color>]
[FIELDS <field 1> [<field option list 1>] |
 <calculated field 1> = <exp 1> [<calculated field option list 1>]
 [, <field 2> [<field option list 2>] |
 <calculated field 2> = <exp 2> [<calculated field option list 2>]...]]
[FREEZE <field 3>]
[LOCK <expN 1>]
[NOAPPEND]
[NOEDIT | NOMODIFY]

COLOR <color>

Specifies the color of the cells in the BROWSE. The current highlighted cell has its own, fixed color. The <color> is made up of a foreground color and a background color, separated by a forward slash (/). You may use a Windows-named color, one of the basic 16-color color codes, or a user-defined color name. For more information on colors, see *colorNormal*.

FIELDS <field 1> [<field option list 1>] |
<calculated field 1> = <exp 1> [<calculated field option list 1>]
[, <field 2> [<field option list 2>] |
<calculated field 2> = <exp 2> [<calculated field option list 2>] ...]]

Displays the specified fields, in the order they're listed, in the Table window. Options for <field option list 1>, <field option list 2>, which apply to <field 1>, <field 2>, and so on, affect the way these fields are displayed. These options are as follows:

Option	Description
\<column width>	The width of the column within which <field 1> appears when <field 1> is character type
\B = <exp 1>, <exp 2>	RANGE option; forces any value entered in <field 1> to fall within <exp 1> and <exp 2>, inclusive.
\C =<color>	COLOR option; sets the foreground and/or background colors of the column according to the values specified in <color>
\H = <expC>	HEADER option; causes <expC> to appear above the field column in the Table window, replacing the field name
\P = <expC>	PICTURE option; displays <field 1> according to the PICTURE or FUNCTION clause <expC>
\V = <condition> [\E = <expC>]	VALID option; allows a new <field 1> value to be entered only when <condition> evaluates to *true*

ERROR MESSAGE option; \E = <expC> causes <expC> to appear when <condition> evaluates to *false*

Note

You may also use the forward slash (/) instead of the backslash (\) when specifying only a single option in a field option list.

Read-only calculated fields are composed of an assigned field name and an expression that results in the calculated field value, for example:

```
browse fields commission = RATE * SALEPRICE
```

Options for calculated fields affect the way these fields are displayed. These options are as follows:

Option	Description
\<column width>	The width of the column within which <calculated field 1> is displayed
\H = <expC>	Causes <expC> to appear above the calculated field column in the Table window, replacing the calculated field name

FREEZE <field 3>

Restricts editing to <field 3>, although other fields are visible.

LOCK <expN 2>

Keeps the first <expN 2> fields in place onscreen as you move the cursor to fields on the right.

NOAPPEND

Prevents records from being added when you cursor down past the last record in the Table window. The NOAPPEND option works in dBASE versions up to, and including, 5.7. It has no affect in dBASE versions higher than 5.7, or in *dBASE Plus*.

NOEDIT | NOMODIFY

Prevents you from modifying records from the Table window. The NOEDIT | NOMODIFY option works in dBASE versions up to, and including, 5.7. It has no affect in dBASE versions higher than 5.7, or in *dBASE Plus*.

Description

The BROWSE command opens a table grid in a window, displaying the fields in the currently selected work area. It is intended more for interactive use; in an application, you have more control over a Browse or Grid object in a form.

The BROWSE command is modeless. After the window is opened, the next statement is executed.

OODML

Use a Grid control on a form.

CALCULATE

Topic group Related topics

Performs financial and statistical operations for values of records in the current table.

Syntax

CALCULATE <function list>
[<scope>]
[FOR <condition 1>]

[WHILE <condition 2>]
[TO <memvar list> | TO ARRAY <array>]

<function list>

You can use one or more of the following functions:

Function	Purpose
AVG(<expN>)	Calculates the average of the specified numeric expression.
CNT( )	Counts the number of records in the current table.
MAX(<expC> | <expN> | <expD>)	Calculates the maximum value of the specified numeric, character, or date expression.
MIN(<expC> | <expN> | <expD>)	Calculates the minimum value of the specified numeric, character, or date expression.
NPV(<expN 1>, <expN 2> [, <expN 3>])	Calculates the net present value of the numeric values in <expN 2>; <expN 1> is the periodic interest rate, expressed as a decimal; <expN 3> is the initial investment and is generally a negative number.
STD(<expN>)	Calculates the standard deviation of the specified numeric expression.
SUM(<expN>)	Calculates the sum of the specified numeric expression.
VAR(<expN>)	Calculates the variance of the specified numeric expression.

<scope>
FOR <condition 1>
WHILE <condition 2>

The scope of the command. The default scope is ALL.

TO <memvar list> | TO ARRAY <array>

Initializes and stores the results to the variables (or properties) of <memvar list> or stores results to the existing array <array>. <array> can be a single- or multidimensional array; the array elements are accessed via their element numbers, not their subscripts.

Description

CALCULATE uses one or more of the eight associated functions listed in the previous table to calculate and store sums, maximums, minimums, averages, variances, standard deviations, or net present values of specified expressions. The expressions are usually, but not required to be, based on fields in the current table. You can calculate values in a work area other than the current work area if you set a relation between the work areas.

CALCULATE can also return the count or number of records in the current table. These special functions, with the exception of MAX( ) and MIN( ), can be used only with CALCULATE.

CALCULATE can use the same function on different expressions or different functions on the same expression. For instance, if your table contains a Salary field and a Bonus field, you can issue the command:

```
calculate sum(SALARY), sum(BONUS), avg(SALARY), avg(12 * (SALARY + BONUS))
```

CALCULATE stores results to variables or to an existing array in the order of the specified functions. If you store the results to memory variables, specify the same number of variables as the number of functions in the CALCULATE command line. If you store the values in an array, the array must already exist, and the array must contain at least as many elements as the number calculations.

If SET TALK is ON, CALCULATE displays the results in the result pane of the Command window. The SET DECIMALS setting determines the number of decimal places that CALCULATE displays.

CALCULATE treats a blank numeric field as containing 0 and includes the field in its calculations. For example, if you calculate the average of a numeric field in a table containing ten records, five of which are blank, CALCULATE divides the sum by 10 to find the average. Furthermore, if you calculate the minimum of the same table field and five records contain positive non-zero numbers and the five others are blank in the same fields, CALCULATE returns 0 as the minimum. If you want to exclude blank fields when using CALCULATE, be sure to specify a condition such as FOR .NOT. ISBLANK(numfield).

When calculating an empty column, CALCULATE works differently on level 7 tables than it does on table levels less than 7. With level 7 tables, CALCULATE returns "null" on an empty column. For an empty column in tables with a level less than 7, CALCULATE returns 0.

Although you can use the SUM or AVERAGE commands to find sums and averages, if you are mixing sums and averages, CALCULATE is faster because it runs through the table just once while making all specified calculations.

OODML

Loop through the rowset to calculate the values.

CHANGE( )

Topic group Related topics

Returns *true* if another user has changed a record since it was read from disk.

Syntax

CHANGE([<alias>])

<alias>

The work area you want to check.

Description

Use CHANGE( ) to determine if another user has made changes to a record since it was read from disk. If the record has been changed, you might want to display a message to the user before allowing the user to continue.

Note

CHANGE( ) only works with DBF tables.

For CHANGE( ) to return information, the table being checked must have a _DBASELOCK field. Use CONVERT to add a _DBASELOCK field to a table. If the table doesn't contain a _DBASELOCK field, CHANGE( ) returns *false*

CHANGE( ) compares the counter in the workstation's memory image of _DBASELOCK to the counter stored on disk. If they are different, the record has changed, and CHANGE( ) returns *true*.

You can reset the value of CHANGE( ) to *false* by moving the record pointer. GOTO BOOKMARK( ) rereads the current record's _DBASELOCK field, and a subsequent CHANGE( ) returns *false*, unless another user has changed the record in the interim between moving to it and issuing CHANGE( ).

OODML

Call rowset.fields["_DBASELOCK"].update

CLEAR AUTOMEM

Topic group Related topics

Initializes automem variables with empty values for the current table.

Syntax

CLEAR AUTOMEM

Description

Use CLEAR AUTOMEM to initialize a set of automem variables containing empty values for the current table. CLEAR AUTOMEM creates any automem variables that don't exist already. If the variables exist, CLEAR AUTOMEM reinitializes them. If no table is in use, CLEAR AUTOMEM doesn't create any variables.

CLEAR AUTOMEM creates normal variables. They default to private scope when CLEAR AUTOMEM is executed in a program or function. If there is a danger of overwriting previously created public or private variables with the same name, you must declare the new automem variables PRIVATE individually by name before issuing CLEAR AUTOMEM, just as you would if you created the variables manually.

Automem variables have the same names and data types as the fields in an active table. You can create empty automem variables automatically for the current table by using CLEAR AUTOMEM or USE...AUTOMEM, or manually by using STORE or the assignment operators.

OODML

The Rowset object contains an array of Field objects, accessed through its *fields* property. These Field objects have *value* properties that may be programmed like variables.

CLEAR FIELDS

Topic group Related topics

Removes the fields list defined with the SET FIELDS TO command.

Syntax

CLEAR FIELDS

Description

Use CLEAR FIELDS to remove the SET FIELDS TO <field list> setting in all work areas and automatically turn SET FIELDS to OFF, thus making all fields in all open tables accessible. You can use CLEAR FIELDS prior to specifying a new fields list with SET FIELDS TO. You might also want to use CLEAR FIELDS at the end of a program. CLEAR FIELDS has the same effect as SET FIELDS TO with no options.

OODML

No direct equivalent. When accessing the *fields* array, you may include program logic to include or exclude specific fields.

CLOSE DATABASES

Topic group Related topics

Closes databases, including their tables and indexes.

Syntax

CLOSE DATABASES [<database name list>]

<database name list>

The list of database names, separated by commas. If no list is specified, all open databases are closed.

Description

Closing a database closes all the open tables in the database, including all the index, memo, and other associated files. For the default database, which gives access to DBF and DB tables, this means all open tables in all work areas.

CLOSE DATABASES only closes those tables opened in the current workset. For more information on worksets, see CREATE SESSION.

OODML

Set the *active* property of the Database object (or all its Query objects) to *false*.

CLOSE INDEXES

Topic group Related topics

Closes DBF index files in the current work area.

Syntax

CLOSE INDEXES

Description

Closes index (.MDX and .NDX) files open in the current work area. This option does not close the production .MDX file.

OODML

Clear the *indexName* property of the Rowset object.

CLOSE TABLES

Topic group Related topics

Closes all tables.

Syntax

CLOSE TABLES

Description

Closes all tables in all work areas or all tables in the current database, if one is selected.

CLOSE TABLES only closes those tables opened in the current workset. For more information on worksets, see CREATE SESSION.

OODML

Set the *active* property of all the Query objects to *false*.

COMMIT( )

Topic group Related topics

Clears the transaction log, committing all logged changes.

Syntax

COMMIT([<database name expC>])

<database name expC>

The name of the database in which to complete the transaction.

If you began the transaction with BEGINTRANS(<database name expC>), you must issue COMMIT(<database name expC>). If instead you issue COMMIT( ), *dBASE Plus* ignores the COMMIT( ) statement.

If you began the transaction with BEGINTRANS( ), <database name expC> is an optional COMMIT( ) argument. If you include it, it must refer to the same database as the SET DATABASE TO statement that preceded BEGINTRANS( ).

Description

A transaction works by logging all changes. If an error occurs while attempting one of the changes, or the changes need to be undone for some other reason, the transaction is canceled by calling ROLLBACK( ). Otherwise, COMMIT( ) is called to clear the transaction log, thereby indicating that all the changes in the transaction were committed and that the transaction as a whole was posted.

For more information on transactions, see BEGINTRANS( ).

OODML

Call the *commit( )* method of the Database object.

CONTINUE

Topic group Related topics

Continues a search for the next record that meets the conditions specified in a previously issued LOCATE command.

Syntax

CONTINUE

Description

CONTINUE continues the search of the last LOCATE issued in the selected work area. When you issue the LOCATE command, the current table is searched sequentially for the first record that matches the search criteria.

If a record is found, the record pointer is left at the matching record. To continue the search, issue the CONTINUE command. Whenever a match is found, FOUND( ) returns *true*. If match is not found, the record pointer is left after the last record checked, which usually leaves it at the end-of-file. Also, FOUND( ) returns *false*.

If SET TALK is ON, CONTINUE will display the record number of the matching record in the result pane of the Command window if you are searching a DBF table. If no match is found, CONTINUE will display "End of Locate scope".

If you issue CONTINUE without first issuing a LOCATE command for the current table, an error occurs..

OODML

Use the Rowset object's *locateNext( )* method. This method also allows going backwards or to the nth match.

COPY

Topic group Related topics

Copies records from the current table to another table or text file.

Syntax

```
COPY TO <filename>
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]
[FIELDS <field list>]
[[TYPE] DBASE | DBMEMO3 | PARADOX | SDF |
  DELIMITED [WITH
    <char> | BLANK]] |
[[WITH] PRODUCTION]
```

TO <filename>

Specifies the name of the table or file you want to create.

<scope>
FOR <condition 1>
WHILE <condition 2>

The scope of the command. The default scope is ALL.

FIELDS <field list>

Specifies which fields to copy to the new table.

[TYPE] DBASE | DBMEMO3 | PARADOX | SDF |
DELIMITED [WITH <char> | BLANK]

Specifies the format of the file to which you want to copy data. The TYPE keyword is included for readability only; it has no effect on the operation of the command. The following table provides a description of the different file formats that are supported:

Type	Description
DBASE	A dBASE table. If you don't include an extension for <filename>, *dBASE Plus* assumes a .DBF extension.
DBMEMO3	Table (.DBF) and memo (.DBT) files in dBASE III PLUS format.
PARADOX	A Paradox table. If you don't include an extension for <filename>, *dBASE Plus* assumes a .DB extension.
SDF	A System Data Format text file. Records in an SDF file are fixed-length, and the end of a record is marked with a carriage return and a linefeed. If you don't specify an extension, *dBASE Plus* assumes .TXT.
DELIMITED	A text file with fields separated by commas. These files are also referred to as CSV (Comma Separated Value) files. Character fields are delimited with double quotation marks when they are not empty() or null. Each carriage return and linefeed indicates a new record. If you don't specify an extension, *dBASE Plus* assumes .TXT.
DELIMITED WITH <char>	Indicates that character data is delimited with the character <char> instead of with double quotes. For example, if delimited with a single quote instead of a double quote, the clause would be: DELIMITED WITH '
DELIMITED	Indicates that data is separated with spaces instead of commas, with no delimiters.

WITH BLANK

[WITH] PRODUCTION

Specifies copying the production .MDX file along with the associated table. This option can be used only when copying to another dBASE table.

Description

Use COPY to copy all or part of a table to a file of the same or a different type. If an index is active, COPY arranges the records of the new table or file according to the indexed order.

The COPY command does not copy a _DBASELOCK field in a table that you've created with CONVERT.

The COPY command does not copy standard, custom or referential integrity properties to the new file. Standard properties include default, maximum, minimum and required.

COPY TO [WITH] PRODUCTION results in a table whose natural order mimics that of the active index being copied.

COPY TO [WITH] PRODUCTION also changes the "date of last update" (datestamp) in the headers of newly created files, to reflect the date they were created (in other words, today's date).

Use the COPY TABLE command to make a copy of a table, including all its index, memo, and other associated files, if any. Unlike the COPY command, the table does not have to be open, and an exact copy of all the records is always made. COPY TABLE copies all field property information and, unlike COPY TO [WITH] PRODUCTION, does not change the datestamp in headers of newly created .dbf and .mdx files

When COPYing to text files, SDF or DELIMITED, non-character fields are written as follows:

Numbers are written as-is.
Logical or boolean fields use the letter T for *true* and F for *false*.
Dates are written in the format YYYYMMDD.

If you COPY a table containing a memo field to another dBASE table, *dBASE Plus* creates another file with the same name as the table but having a .DBT extension, and copies the contents of the memo field to it. If, however, you use the SDF or DELIMITED options and COPY to a text file, *dBASE Plus* doesn't copy the memo fields.

Deleted records are copied to the target file (if it's a dBASE table) unless a FOR or WHILE condition excludes them or unless SET DELETED is ON. Deleted records remain marked for deletion in the target dBASE table.

You can use COPY to create a file containing fields from more than one table. To do that, open the source tables in different work areas and define a relation between the tables. Use SET FIELDS TO to select the fields from each table that you want to copy to a new file. Before you issue the COPY command, SET FIELDS must be ON and you must be in the work area in which the parent table resides.

The COPY command does not verify that the files you build are compatible with other software programs. You may specify field lengths, record lengths, number of fields, or number of records that are incompatible with other software. Check the file limitations of your other software program before exporting tables using COPY.

OODML

Use the UpdateSet object's *copy( )* method. Set filter options in the *source* rowset.

COPY BINARY

Topic group Related topics

Copies the contents of the specified binary field to a file.

Syntax

COPY BINARY <field name> TO <filename>
[ADDITIVE]

<field name>

The binary field to copy.

TO <filename>

The name of the file where the contents of the binary field are copied. For predefined binary file types, *dBASE Plus* assigns the appropriate extension, for example, .BMP, .WAV, and so on. For user-defined binary type fields, *dBASE Plus* assigns a .TXT extension by default.

ADDITIVE

Appends the contents of the binary field to the end of an existing file. Without the ADDITIVE option, *dBASE Plus* overwrites the previous contents of the file.

Description

Use COPY BINARY to export data from a binary field in the current record to a file. You can use binary fields to store text, images, sound, video, and other user-defined binary data.

If you specify the ADDITIVE option, *dBASE Plus* appends the contents of the binary field to the end of the named file, which lets you combine the contents of binary fields from more than one record. When you don't use ADDITIVE, *dBASE Plus* displays a warning message before overwriting an existing file if SET SAFETY is ON. Note that you can't combine the data from more than one field for many of the predefined binary data types. For example, you can store only a single image in a binary field or file, so do not use the ADDITIVE option of COPY BINARY when copying an image to a file.

OODML

Use the Field object's *copyToFile( )* method.

COPY MEMO

Topic group Related topics

Copies the contents of the specified memo field to a file.

Syntax

COPY MEMO <memo field> TO <filename>
[ADDITIVE]

<memo field>

The memo field to copy.

TO <filename> | ?

The name of the text file where text will be copied. The default extension is .TXT.

ADDITIVE

Appends the contents of the memo field to the end of an existing text file. Without the ADDITIVE option, *dBASE Plus* overwrites any previous text in the text file.

Description

Use COPY MEMO to export memo file text in the current record to a text file. You can also use COPY MEMO to copy images or other binary-type data to a file; however, binary fields are recommended for storing images, sound, and other user-defined binary information.

If you specify the ADDITIVE option, *dBASE Plus* appends the contents of the memo field to the end of the named file, which lets you combine the contents of memo fields from more than one record. When you don't use ADDITIVE, *dBASE Plus* displays a warning message before overwriting an existing file if SET SAFETY is ON. You can store only a single image in either a memo field or in a file, so do not use the ADDITIVE option of COPY MEMO when copying an image to a file. (RESTORE IMAGE can display an image stored in either a memo field or a text file.)

OODML

Use the Field object's *copyToFile( )* method.

COPY STRUCTURE

Topic group Related topics

Creates an empty table with the same structure as the current table.

Syntax

COPY STRUCTURE TO <filename>
[[TYPE] PARADOX | DBASE]
[FIELDS <field list>]
[[WITH] PRODUCTION]

<filename>

The name of the table you want to create.

[TYPE] PARADOX | DBASE

Specifies the type of table you want to create, if <filename> does not include an extension. This option overrides the current SET DBTYPE setting. The TYPE keyword is included for readability only; it has no effect on the operation of the command.

FIELDS <field list>

Determines which fields *dBASE Plus* includes in the structure of the new table. The fields appear in the order specified by <field list>.

[WITH] PRODUCTION

Creates a production .MDX file for the new table. The new index file has the same index tags as the production index file associated with the original table.

Description

The COPY STRUCTURE command copies the structure of the current table but does not copy any records. If SET SAFETY is OFF, *dBASE Plus* overwrites any existing tables of the same name without issuing a warning message.

The COPY STRUCTURE command copies the entire table structure unless limited by the FIELDS option or the SET FIELDS command. When you issue COPY STRUCTURE without the FIELDS <field list> option, *dBASE Plus* copies the fields in the SET FIELDS TO list to the new table. The _DBASELOCK field created with the CONVERT command is not copied to new tables.

You can use COPY STRUCTURE to create an empty table structure with fields from more than one table. To do so,

1. Open the source tables in different work areas.
   - Use the FIELDS <field list> option, including the table alias for each field name not in the current table.

OODML

No equivalent.

COPY STRUCTURE EXTENDED

Topic group Related topics

Creates a new table whose records contain the structure of the current table.

Syntax

COPY STRUCTURE EXTENDED TO <filename>
[[TYPE] PARADOX | DBASE]

or

COPY TO <filename>
STRUCTURE EXTENDED
[[TYPE] PARADOX | DBASE]

<filename>

The name of the table that you want to create to contain the structure of the current table.

[TYPE] PARADOX | DBASE

Specifies the type of table you want to create, if <filename> does not include an extension. This option overrides the current SET DBTYPE setting. The TYPE keyword is included for readability only; it has no effect on the operation of the command.

Description

COPY STRUCTURE EXTENDED copies the structure of the current table to records in a new table.

COPY STRUCTURE EXTENDED first defines a table, called a structure-extended table, containing five fields of fixed names, types, and lengths. Once the structure-extended table is defined, COPY STRUCTURE EXTENDED appends records that provide information about each field in the current table. The fields in the structure-extended table store the following information about fields in the current table:

Field	Contents
FIELD_NAME	Character field that contains the name of the field.
FIELD_TYPE	Character field that contains the field's data type.
FIELD_LEN	Numeric field that contains the field length.
FIELD_DEC	Numeric field that contains the number of decimal places for numeric fields.
FIELD_IDX	Character field that indicates if index tags were created on individual fields in the table.

When the process is complete, the structure-extended table contains as many records as there are fields in the current table. You can then use CREATE...FROM to create a new table from the information provided by the structure-extended table.

No record is created in the structured-extended table for the _DBASELOCK field created with the CONVERT command.

OODML

Use the Database object's *executeSQL*( ) method to call the SQL command CREATE TABLE (see CREATE STUCTURE EXTENDED) to create the structure-extended table. Then use a loop to populate the table with information from the array of Field objects.

COPY TABLE

Topic group Related topics

Makes a copy of a table.

Syntax

COPY TABLE <source tablename> TO <target tablename>
[[TYPE] PARADOX | DBASE]

<source table name>

The name of the table that you want to copy. You can also copy a table in a database (defined using the BDE Administrator) by specifying the database as a prefix (enclosed in colons) to the name of the table, that is, :database name:table name. If the database is not already open, *dBASE Plus* displays a dialog box in which you specify the parameters, such as a login name and password, necessary to establish a connection to that database.

<target table name>

The name of the table you want to create. The table type is the same as the source table. If you copy a table in a database, you must specify the same database as the destination of the target table.

[TYPE] PARADOX | DBASE

Specifies the default extension for the both the source table and target table: .DB for Paradox and .DBF for dBASE. This overrides the current setting of DBTYPE. You cannot change the table type during the copy; this clause is useful only when using filenames that do not have extensions.

Description

Use the COPY TABLE command to make a copy of a table, including all its index, memo, and other associated files, if any. Unlike the COPY command, the table does not have to be open, and an exact copy of all the records is always made.

OODML

Use the Database object's *copyTable*( ) method.

COPY TO ARRAY

Topic group Related topics Example

Copies data from non-memo fields of the current table, overwrites elements of an existing array, and moves the record pointer to the last record copied.

Syntax

COPY TO ARRAY <array>
[<scope>]

[FOR <condition 1>]
[WHILE <condition 2>]
[FIELDS <field list>]

<array>

A reference to the target array

<scope>
FOR <condition 1>
WHILE <condition 2>

The scope of the command. The default scope is ALL, until <array> is filled.

FIELDS <field list>

Copies data from the fields in <field list> in the order of <field list>. Without FIELDS, *dBASE Plus* copies all the fields the array can hold in the order they occur in the current table.

Description

Use COPY TO ARRAY to copy records from the current table to an existing array. COPY TO ARRAY treats the columns in a one-dimensional array like a single record of fields; and treats a two-dimensional array like a table, with the rows (the first dimension) of the array like records, and the columns (the second dimension) like fields.

To copy the fields from a single record, create a one-dimensional array the same size as the number of fields to copy. To copy all the fields in the record, use FLDCOUNT( ) to get the number of fields; for example

```
a = new Array( fldcount( ) )
```

To copy multiple records, create a two-dimensional array. The first dimension will indicate the number of records. The second dimension indicates the maximum number of fields. To copy all the records, use RECCOUNT( ) to get the number of records; for example

```
a = new Array( reccount( ), fldcount( ) )
```

If the array has more columns than the table has fields, the additional elements will be left untouched. Similarly, if a two-dimensional array has more rows than the table, the additional rows are left untouched.

COPY TO ARRAY does not copy memo (or binary) fields; these fields should not be counted when sizing the target array.

COPY TO ARRAY copies records in their current order and, within each record, in field order unless you use the FIELDS option to specify the order of the fields to copy.

After copying, the record pointer is left at the last record copied, unless the array has more rows than the table has records. In this case, the record pointer is left at the end-of-file.

OODML

Use two nested loops, the first to traverse the rowset, and the second to copy the *value* properties of the Field objects in the rowset's *fields* array to the target array's elements.

COUNT

Topic group Related topics

Counts the number of records that match specified conditions.

Syntax

COUNT
[<scope>]

[FOR <condition 1>]
[WHILE <condition 2>]
[TO <memvar>]

<scope>
FOR <condition 1>
WHILE <condition 2>

The scope of the command. The default scope is ALL.

TO <memvar>

Stores the result of COUNT, a number, to the specified variable (or property).

Description

Use COUNT to total the number of visible records. The current index, filter, key constraints, DELETED setting, and other factors control which records are visible at any time. You may specify further criteria with the <scope> and FOR and WHILE conditons.

If the COUNT is not stored to a memvar, the result is displayed in a dialog box. If the COUNT is stored to a memvar and SET TALK is ON, the result is also displayed in the status bar.

COUNT automatically locks the table during its operation if SET LOCK is ON (the default), and unlocks it after the count is finished. If SET LOCK is OFF, you can still perform a count; however the result may change if another user changes the table.

You can also count the total number of records in a table using the RECCOUNT( ) function. However, unlike COUNT, RECCOUNT( ) does not let you specify conditions to qualify the records it counts.

OODML

Use the Rowset object's *count( )* method.

CREATE SESSION

Topic group Example

Creates a new session—now referred to as a workset—and immediately selects it.

Syntax

CREATE SESSION

Description

Use CREATE SESSION in an application that uses form-based data handling and the Xbase DML. Applications that only use the data objects generally do not need CREATE SESSION.

A workset is the more precise term for what was called a session in earlier versions of *Visual* dBASE and is used to encapsulate separate user tasks. It consists of the set of all 225 work areas and the current settings of most of the SET commands. There is always an active workset. When *dBASE Plus* starts, the settings are read from the PLUS.ini file and all work areas are empty. This is sometimes referred to as the startup workset.

Whenever you open or close a table or change a setting, that occurs in the current workset. Commands that affect all work areas, like CLOSE DATABASES, affect all work areas in the current workset only. Record locks are workset-based. If a record is locked in one workset, you cannot lock that same record from another workset; but you could lock that record if the same table is open in another work area in the same workset.

When you issue CREATE SESSION, a new workset is created and made active. A new unused set of work areas is created and all settings are reread from the .INI file. Any previously existing worksets are unaffected, except that they are no longer active. In fact, you cannot change anything about a dormant workset; you must make it active first.

Whenever a form is created, it is bound to the currently active workset. Any number of forms may be bound to a single workset. Each workset has a reference count that indicates the number of forms bound to it. The Command window and Navigator are both bound to the startup workset.

Whenever a form receives focus or any of its methods are called, its workset is activated. This means that all commands, functions, and methods take place in the context of a specific workset and have no effect on the tables or settings in other worksets.

Note

Worksets have no effect on variables.

When a form is released (either explicitly or when there are no more references to the form) its workset's reference count is reduced by one. If that reduces the reference count to zero, the workset is also released.

Whenever a workset is released, any tables that are open in it are closed automatically.

The active workset's reference count is also checked:

Just before another workset is activated (usually by giving focus to a form in another workset)
Whenever CREATE SESSION is executed (before the new workset is created)
When a form method has finished executing.

If the count is zero, the active workset is released. When a form method is finished, it also selects the workset that was active when the method started. So if you click a button button on a form that currently does not have focus, and that button's *onClick* event handler (all event handlers are methods) has a CREATE SESSION command then the sequence of events is as follows:

1. Clicking the form causes a focus change. The active workset is checked; if its reference count is zero, it is released.
    - The form's workset is activated.
    - The *onClick* executes, creating and activating a new workset.
    - The *onClick* ends. If the reference count of the just-created workset is zero, which it would be if the method didn't create any forms after the CREATE SESSION, it is released.
    - The form's workset, the one that was active when the method was executed, is reactivated. It is now the active workset.

Clicking the button again would only go through steps 3 through 5, because the form still has focus, so there is no focus change.

OODML

Use Session objects.

CREATE...FROM

Topic group Related topics

Creates a table with the structure defined by using the COPY STRUCTURE EXTENDED or CREATE...STRUCTURE EXTENDED commands.

Syntax

```
CREATE <filename 1>
 [[TYPE] PARADOX | DBASE]
FROM <filename 2>
  [[TYPE] PARADOX | DBASE]
```

<filename 1>

The name of the table you want to create.

[TYPE] PARADOX | DBASE

Specifies the type of table you want to create, if <filename> does not include an extension. This option overrides the current SET DBTYPE setting. The TYPE keyword is included for readability only; it has no effect on the operation of the command.

FROM <filename 2>
[TYPE] PARADOX | DBASE

Identifies the table that contains the structure of the table you want to create.

Description

The CREATE...FROM command is most often used with the COPY STRUCTURE EXTENDED command in a program to create a new table from another table that defines its structure, instead of using the interactive CREATE or MODIFY STRUCTURE commands. To do this, you can

1. Use COPY STRUCTURE EXTENDED to create a table whose records provide information on each field of the original table.
    - Optionally, modify the structural data in the new table with any dBASE command used to manipulate data, such as REPLACE.
    - Use CREATE...FROM to create a new table from the structural information in the structure extended file. The new table is active when you exit CREATE...FROM.

The table created with CREATE...FROM becomes the current table in the currently selected work area. If the CREATE...FROM operation fails for any reason, no table remains open in the current work area.

If any fields in the table created with COPY STRUCTURE EXTENDED have index flag fields set, CREATE...FROM also creates a production .MDX file with the specified index tags.

OODML

No equivalent.

CREATE...STRUCTURE EXTENDED

Topic group Related topics

Creates and opens a table that you can use to design the structure of a new table.

Syntax

```
CREATE <filename> STRUCTURE EXTENDED
[[TYPE] PARADOX | DBASE]
```

<tablename> | ?

The name of the table you want to create.

[TYPE] PARADOX | DBASE

Specifies the type of table you want to create, if <filename> does not include an extension. This option overrides the current SET DBTYPE setting. The TYPE keyword is included for readability only; it has no effect on the operation of the command.

Description

CREATE...STRUCTURE EXTENDED creates an empty table, called a structure-extended table, containing five fields of fixed names, types, and lengths. The fields correspond to attributes that describe fields in the table you want to create:

Field	Contents
FIELD_NAME	Character field that contains the name of the field.
FIELD_TYPE	Character field that contains the field's data type.
FIELD_LEN	Numeric field that contains the field length.
FIELD_DEC	Numeric field that contains the number of decimal places for numeric fields.
FIELD_IDX	Character field that indicates if index tags were created on individual fields in the table.

The CREATE...STRUCTURE EXTENDED command is similar to the COPY STRUCTURE EXTENDED command. However, unlike COPY STRUCTURE EXTENDED, which creates a table with records providing information on fields in the current table, CREATE...STRUCTURE EXTENDED creates an empty structure-extended table. After using CREATE...STRUCTURE EXTENDED to create a new table, add records to define the structure of a new table. Then use the CREATE...FROM command to create a new table from the field definitions stored in the structure-extended table.

OODML

Use the SQL command CREATE TABLE to create the STRUCTURE EXTENDED table.

DATABASE( )

Topic group Related topics

Returns the name of the current database from which tables are accessed.

Syntax

DATABASE( )

Description

DATABASE( ) returns the name of the current default database selected with the SET DATABASE command. If no database is open, the DATABASE( ) function returns an empty string ("").

Note: Databases are defined with the BDE Administrator.

OODML

Check the Database object's *databaseName* property.

DBF( )

Topic group Related topics

Returns the name of a table open in the current or a specified work area.

Syntax

DBF([<alias>])

<alias>

The work area to check.

Description

DBF( ) returns the name of the table open in a specified work area. If the table is a file on disk, as it is with DBF and DB tables, the filename includes the extension and the drive letter. If SET FULLPATH is ON, the DBF( ) function also returns the directory location of the table in addition to the table name.

If no table is in use in the current or specified work area, DBF( ) returns an empty string ("").

OODML

Check the Rowset object's *tableName* property.

DELETE

Topic group Related topics Example

Deletes records from the current table.

Syntax

DELETE
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]

<scope>
FOR <condition 1>
WHILE <condition 2>

The scope of the command. The default scope is NEXT 1, the current record only.

Description

DBF tables support the concept of soft deletes, where the record is marked as deleted and normally hidden when SET DELETED is ON (the default). If you SET DELETED OFF, you can see the deleted records along with the records that are not marked as deleted. You can RECALL the record to undelete it. To actually remove the record from the table, you must PACK the table. If you use the LIST or DISPLAY commands to display records, records marked as deleted are displayed with an asterisk.

For other table types, when you delete a record, it is removed from the table and cannot be recovered. (Some tables still require you to perform a maintenance operation on the table to reclaim the unused space. For more information, refer to your database server documentation.)

Relying on soft deletes to be able to recover information from deleted records is not recommended. This technique does not scale well to other databases, because they don't support soft deletes. If you want to make data available for recover, consider using an identically-structured purge table that stores copies of the records that you have deleted.

Soft deletes are useful when you want to recycle deleted records. This obviates the need to PACK the table. You BLANK the record before you DELETE it. Then whenever you need to add a new record, you can search for a deleted record and reuse it.

To delete all records from a table, use ZAP.

OODML

Use the Rowset object's *delete*( ) method. There is no support for soft deletes; if you *delete*( ) a row in a DBF table, there is no corresponding dBL method that will recall it. You may still use the RECALL command.

DELETE TABLE

Topic group Related topics

Deletes a specified table.

Syntax

DELETE TABLE <filename> [[TYPE] PARADOX | DBASE]

<filename>

The name of the table that you want to delete.

[TYPE] PARADOX | DBASE

Specifies the type of table you want to delete, if <filename> does not include an extension. This option overrides the current SET DBTYPE setting. The TYPE keyword is included for readability only; it has no effect on the operation of the command.

Description

Use the DELETE TABLE command to delete a table and its index, memo, and other associated files. Make sure the table is not in use before you attempt to delete it.

OODML

Use the Database object's *dropTable*( ) method.

DELETE TAG

Topic group Related topics Example

Deletes index tags from tables.

Syntax

```
DELETE TAG <tag name 1>
  [OF <filename 1>]
[, <tag name 2>
  [OF <filename 2>]...]
```

<tag name 1>, <tag name 2>, ... <tag name n>

The index tag names to delete.

OF <filename 1> | ? | <filename skeleton 1>

For DBF tables, specifies the .MDX file containing the tag name to delete. If you specify a file without including an extension, *dBASE Plus* assumes an .MDX extension. If you don't specify an index file, *dBASE Plus* assumes the index tag you want to delete is in the index file with the same name as the current table.

Description

Use DELETE TAG to delete index tags from .MDX files for dBASE tables or secondary indexes on a Paradox table. *dBASE Plus* allows a maximum of 47 index tags in a single .MDX file, so deleting unneeded tags frees slots for new tags as well as reducing the amount of disk space and memory that an .MDX file requires.

For dBASE tables, the .MDX file must be open when you delete the tags. If you delete all tags in an .MDX file, the .MDX file is also deleted. If you delete the production .MDX file by deleting all index tags, the table file header is updated to indicate there is no longer a production index associated with the table.

The table associated with the indexes you want to delete must be opened in exclusive mode. When accessing a Paradox table, specifying DELETE TAG without an argument deletes the primary index.

OODML

Use the Database object's *dropIndex( )* method.

DELETED( )

Topic group Related topics

Indicates if the current record is marked as deleted.

Syntax

DELETED([<alias>])

<alias>

A work area to check.

Description

DELETED( ) returns *true* if the current record in the specified work area is marked as deleted otherwise, DELETED( ) returns *false*.

If no table is open in the current or specified work area, DELETED( ) also returns *false*.

OODML

No support for soft deletes.

DESCENDING( )

Topic group Related topics Example

Indicates if a specified index is in descending order.

Syntax

DESCENDING([<.mdx filename expC>,] [<index position expN> [,<alias>]])

<.mdx filename expC>

The .MDX file that contains the index tag you want to check. The .MDX must be opened in the specified work area. If omitted, all open indexes, including the production .MDX file, are searched.

<index position expN>

the numeric position of the index tag in the specified .MDX file, or the position of the index in the list of open indexes.

<alias>

The work area you want to check.

Note

Unlike most functions, the first parameter is optional. If you omit the first parameter, the remaining parameters shift forward one; the second parameter becomes the first parameter, and so on.

Description

DESCENDING( ) returns *true* if the index tag specified by the <index position expN> parameter was created with the DESCENDING keyword; otherwise, it returns *false*.

The index must be referenced by number. If you do not specify an .MDX file to check, index numbering in the list of open indexes is complicated if you have open .NDX indexes or you have open non-production .MDX files. For more information on index numbering, see SET INDEX. Either way, it is often easier to reference an index tag by name by using the TAGNO( ) function to get the corresponding position number.

If you do not specify an index tag, DESCENDING( ) checks the current master index tag and returns *false* if the master index is an .NDX file or there is no master index.

If the specified .MDX file or index tag does not exist, DESCENDING( ) returns *false*.

OODML

No equivalent

DISPLAY

Topic group Related topics

Displays records from the current table in the result pane of the Command window.

Syntax

DISPLAY
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]
[[FIELDS] <exp list>]
[OFF]
[TO FILE <filename>]
[TO PRINTER]

<scope>
FOR <condition 1>
WHILE <condition 2>

The scope of the command. The default scope is NEXT 1, the current record only.

FIELDS <exp list>

Field names or expressions whose contents (values) you want to display; the names of the fields in the list are separated by commas. If you omit <exp list>, *dBASE Plus* displays all fields in the current table. The FIELDS keyword is included for readability only; it has no affect on the operation of the command.

OFF

Suppresses display of the record number when displaying records from a DBF table.

TO FILE <filename>

Directs output to a file, as well as to the results pane of the Command window. By default, *dBASE Plus* assigns a .TXT extension to <filename>.

TO PRINTER

Directs output to the default printer, as well as to the results pane of the Command window.

Description

Use DISPLAY to view one or more records of the current table in the results pane of the Command window. If SET HEADINGS is OFF, *dBASE Plus* doesn't display field names when you issue DISPLAY. DISPLAY pauses when the results pane is full and displays a dialog box prompting you to display another screenful of information.

Use the TO FILE clause to send the information to a file. Use the TO PRINTER clause to send the information to the printer. In either case, you can use SET CONSOLE OFF to suppress the display of the information in the results pane.

The LIST command is almost identical to DISPLAY, except that:

The default scope for LIST is ALL.

LIST doesn't pause for each screenful of information but rather lists the information continuously. This makes LIST more appropriate when directing output to a file or printer.

Memo fields are displayed as "MEMO" if they contain data or "memo" if they are empty; unless the field is listed in <exp list>, in which case the contents of the memo field is displayed.

OODML

No equivalent

EDIT

Topic group Related topics

Displays fields in the current table for editing.

Syntax

EDIT
[COLUMNAR]
[FIELDS <field1> | <Calc field1> = <Exp1>[<Option for Calc field 1>]
[,<field2> | <Calc field2> = <Exp2>[<Option for Calc field 2>]...]]

COLUMNAR

Creates a form with the field names in one column and the field controls in another column.

[FIELDS <field list>]

Displays the fields specified in <field list> for editing. Field names are separated by commas. The field list may include calculated fields in the format:

```
<calculated field name> = <expression>
```

Description

EDIT displays the current record in the current table in a wizard-generated form for editing.

OODML

Use a Form.

EOF( )

Topic group Related topics Example

Indicates if the record pointer is at the end-of-file.

Syntax

EOF([<alias>])

<alias>

The work area to check.

Description

EOF( ) returns *true* when the record pointer in the current or specified work area is positioned past the last record; otherwise it returns *false*. If you attempt to navigate forward when EOF( ) is *true*, an error occurs.

When you first USE a table, EOF( ) is *true* if the table is empty, or you are using a conditional index with no matching records.

Many operations leave the record pointer at the end-of-file when they are complete or when they fail. For example, EOF( ) returns *true* after SCAN processes the last record in a table, when you use SKIP to pass the last record in a table, when you use LIST with no options, or when SEEK( ) or SEEK fails to find the specified record (and SET NEAR is OFF).

The position at the end-of-file is sometimes referred to as the phantom record. When you get the values of the fields at the phantom record, they are always blank. Attempting to REPLACE field values in the phatom record causes an error.

If no table is open in the specified work area, EOF( ) returns *false*.

OODML

The Rowset object's *endOfSet* property is *true* when the row pointer is past either end of the rowset. Unlike BOF( ) and EOF( ), there is symmetry with the *endOfSet* property. You can determine which end you're on based on the direction of the last navigation.

There is also an *atLast( )* method that determines whether you are on the last row in the rowset, the row before EOF( ).

FDECIMAL( )

Topic group Related topics

Returns the number of decimal places in a specified field of a table.

Syntax

FDECIMAL(<field number expN> [, <alias>])

<field number expN>

The position of the field that you want to evaluate. The first field in a table is field number 1.

<alias>

The work area that contains the field to check.

Description

FDECIMAL( ) returns the number of decimal places in a specified field of a table. FDECIMAL( ) returns zero if the field has no decimal places, if the field is not a numeric field, or if the table doesn't contain a field in the specified position.

OODML

Check the *decimalLength* property of the Field object.

FIELD( )

Topic group Related topics Example

Returns the name of the field in a specified position of a table.

Syntax

FIELD(<field number expN> [, <alias>])

<field number expN>

The position of the field whose name you want returned. The first field in a table is field number 1.

<alias>

The work area to check.

Description

FIELD( ) returns the name of a field in a table based on the specified <field number expN> parameter. The example shows a function that performs the reverse operation, returning the field number for a specified field name.

If the field name has spaces, FIELD( ) returns the name enclosed in colons, for example:

```
:Primary power coupling:
```

FIELD( ) returns an empty string ("") if the table does not contain a field in the specified position.

OODML

Check the *fieldName* property of the Field object.

FLDCOUNT( )

Topic group Related topics

Returns the number of fields in a table. Same as FCOUNT( )

Syntax

FLDCOUNT([<alias>])

<alias>

The work area you want to check.

Description

FLDCOUNT( ) returns the number of fields for the table opened in the current or specified work area. If no table is open, FLDCOUNT( ) returns a value of zero.

OODML

Check the *size* property of Rowset object's *fields* array.

FLDLIST( )

Topic group Related topics

Returns the fields and calculated field expressions of a SET FIELDS TO list.

Syntax

FLDLIST([<field number *expN*>])

<field number expN>

The position of the field or calculated field expression in a SET FIELDS TO list whose name you want returned. If you do not specify a field number, FLDLIST( ) returns the entire field list.

Description

FLDLIST( ) returns the field or calculated field expression in a SET FIELDS TO list that corresponds to a specified field number. If you do not specify a field number, FLDLIST( ) returns the entire field list. Each field name or expression in the field list is separated by a comma. FLDLIST( ) always returns fully-qualified field names, that is, it includes the table or alias name. For read-only fields, FLDLIST( ) appends "/R" to the field name.

FLDLIST( ) returns the field list even if SET FIELDS is OFF. If there is no SET FIELDS TO list, or the specified field number exceeds the number of items in the field list, FLDLIST( ) returns an empty string ("").

OODML

Check the *fieldName* property of the Field object for a normal field. A calculated field is defined by either its *value* property, or by its *beforeGetValue* event.

FLENGTH( )

Topic group Related topics Example

Returns the length of the field in a specified position of a table.

Syntax

FLENGTH(<field number expN> [, <alias>])

<field number expN>

The position of the field whose length you want returned. The first field in a table is field number 1.

<alias>

The work area you want to check.

Description

FLENGTH( ) returns the length of a field in a table based on the specified <field number expN> parameter. The field length for numeric fields includes the decimal digits and one for the decimal point character. Certain field types have fixed lengths. For example, in a DBF table, FLENGTH( ) returns 8 for date fields and 10 for memo fields.

FLENGTH( ) returns 0 if the table does not contain a field in the specified position.

OODML

Check the *length* property of the Field object.

FLOCK( )

Topic group Related topics

Locks a table.

Syntax

FLOCK([<alias>])

<alias>

The work area you want to lock.

Description

Use FLOCK( ) to lock the table in the current work area, or in another specified work area, preventing others from using the table.

When you lock a table with FLOCK( ), only you can make changes to it. However, unlike USE...EXCLUSIVE and SET EXCLUSIVE ON, FLOCK( ) lets other users view the locked table while you are using it. When you lock a table with FLOCK( ), it remains locked until you issue UNLOCK or close the table.

FLOCK( ) is similar to RLOCK( ), except that FLOCK( ) locks an entire table, while RLOCK( ) lets you lock specific records of a table. Use FLOCK( ), therefore, when you need to have sole access to an entire table or related tables—for example, when you need to update multiple tables related by a common key.

FLOCK( ) can lock a table even if another user is viewing data contained in the table. FLOCK( ) is unsuccessful only if another user has explicitly locked the table or a record in the table, or is using a command that automatically locks the table or a record in the table. FLOCK( ) returns *true* if it is successful, and *false* if it is not.

All commands that change table data cause *dBASE Plus* to attempt an automatic record or file lock. If *dBASE Plus* fails to get an automatic record or file lock, it an error occurs. You might want to use FLOCK( ) to handle a lock failure youself, instead of letting the error occur.

When SET REPROCESS is set to 0 (the default) and FLOCK( ) can't immediately lock a table, *dBASE Plus* prompts you to attempt the lock again or cancel the attempt. Until you choose to cancel the function, FLOCK( ) repeatedly attempts to lock the table. Use SET REPROCESS to eliminate being prompted to cancel the FLOCK( ) function, or to set the number of locking attempts.

When you set a relation to a parent table with SET RELATION and then lock the table with FLOCK( ), *dBASE Plus* attempts to lock all child tables. For more information about relating tables, see SET RELATION.

OODML

Use the Rowset object's *lockSet( )* method.

FLUSH

Topic group Related topics

Writes data buffers for the current work area to disk.

Syntax

FLUSH

Description

Use FLUSH to protect data integrity.

When you open a table, *dBASE Plus* loads a certain number of records from that table into a memory buffer, along with the portion of each open index that pertains to those records. When another block of records needs to be read or when you close tables, *dBASE Plus* writes the records in the buffer back to disk, storing any modifications you have made.

FLUSH allows you to save information from the data buffer to disk on-demand, without closing the table. Use FLUSH when you need to store critical information to disk that could otherwise be lost. However, don't use FLUSH too frequently, as it slows execution. For example, in an order-entry application in which only a few orders are entered each hour, FLUSH can save data that might be lost if the power is inadvertently turned off; since orders are entered infrequently, the time needed to execute FLUSH is not important.

OODML

Use the Rowset object's *flush( )* method.

FOR( )

Topic group Related topics Example

Returns the FOR clause of a specified index tag.

Syntax

FOR([<.mdx filename expC>,] [<index position expN> [,<alias>]])

<.mdx filename expC>

The .MDX file that contains the index tag you want to check. The .MDX must be opened in the specified work area. If omitted, all open indexes, including the production .MDX file, are searched.

<index position expN>

the numeric position of the index tag in the specified .MDX file, or the position of the index in the list of open indexes.

<alias>

The work area you want to check.

Note

Unlike most functions, the first parameter is optional. If you omit the first parameter, the remaining parameters shift forward one; the second parameter becomes the first parameter, and so on.

Description

FOR( ) returns a string containing the FOR expression of the specified .MDX tag. FOR( ) returns an empty string ("") if the specified index tag does not have a FOR expression.

The index must be referenced by number. If you do not specify an .MDX file to check, index numbering in the list of open indexes is complicated if you have open .NDX indexes or you have open non-production .MDX files. For more information on index numbering, see SET INDEX. Either way, it is often easier to reference an index tag by name by using the TAGNO( ) function to get the corresponding position number.

If you do not specify an index tag, FOR( ) checks the current master index tag and returns an empty string if the master index is an .NDX file or there is no master index.

If the specified .MDX file or index tag does not exist, FOR( ) returns an empty string.

OODML

No equivalent.

FOUND( )

Topic group Related topics Example

Indicates if the last-issued search command found a match.

Syntax

FOUND([<alias>])

<alias>

The work area you want to check.

Description

FOUND( ) returns *true* if LOCATE, CONTINUE, SEEK, LOOKUP( ), or SEEK( ) found a match in the current or specified table. FOUND( ) returns *false* if no previous search has been performed in that work area, or if the last search was unsuccessful. You can perform searches in different work areas and maintain the status of each FOUND( ) operation, independent of the other work areas.

If tables are linked by a SET RELATION TO command, *dBASE Plus* searches the related tables as you move in the active table with normal navigation or with a search command. This allows you to determine if there is a match in related tables.

When SET NEAR is ON and you use SEEK or SEEK( ),

FOUND( ) returns *true* if an exact match occurs.

FOUND( ) returns *false* for a near match, and the record pointer is moved to the record whose key immediately follows the value searched for.

When SET NEAR is OFF, FOUND( ) returns *false* if a match does not occur.

OODML

Check the return value of the Rowset object's *findKey( )* or *findKeyNearest( )* method.

GENERATE

Topic group Related topics

Adds random records to the current table.

Syntax

GENERATE [<expN>]

<expN>

A number of random-data records to add to the current table. If you specify a <expN> value that is less than or equal to 0, no records are generated. If you don't specify a value for <expN>, *dBASE Plus* prompts you for a number.

Description

GENERATE fills a table with sample data. If a table contains existing records, GENERATE leaves them intact and adds <expN> records to the table.

GENERATE does not create data for memo or binary fields.

OODML

No equivalent.

GO

Topic group Related topics

Moves the record pointer to the specified position in a table.

Syntax

GO[TO]
BOTTOM | TOP | <bookmark> | [RECORD] <expN>
[IN <alias>]

TO

Include for readability only; you may use GO or GOTO.

BOTTOM | TOP | <bookmark> | [RECORD] <expN>

Specifies where to move the record pointer. The following table describes each of the available keywords or options.

Option	Moves the record pointer to
BOTTOM	The last record in the table, using the current index order, if any.
TOP	The first record in the table, using the current index order, if any
<bookmark>	The record saved in <bookmark>
[RECORD] <expN>	That record number. Entering a number in the Command window is equivalent to GO <expN>. The RECORD keyword is included for readability only; it has no affect on the operation of the command.

IN <alias>

The work area where you want to move the record pointer.

Description

GO positions the record pointer in a table.

GO <expN> or GO RECORD <expN> moves the record pointer to a specific record, regardless of whether a master index is open or where that record number occurs in an indexed order. It works only for DBF tables. For tables that do not support record numbers (that is, Paradox and SQL tables), GO <expN> causes an error.

To go to a specific record, use the BOOKMARK( ) to get a bookmark for that record and store it in a variable or property. Then when you need to go back to that record, issue GO <bookmark>.

If an index isn't in use, TOP and BOTTOM refer to the first and last records in a table. If an index is in use for a table, TOP and BOTTOM refer to the first and last records in the index order.

If a relation is set up among several tables, moving the record pointer in the parent table with GOTO repositions the record pointer in a child table to a related record. If there is no related record, the child table record pointer is positioned at the end of the file. Moving the record pointer in a child table, however, doesn't reposition the record pointer in the parent table.

OODML

Use the Rowset object's *first( )*, *last( )*, and *goto( )* methods.

INDEX

Topic group Related topics Example

Creates an index for the current table.

Syntax

For DBF tables:

INDEX ON <key exp>
TAG <tag name>
  [OF <.mdx filename>]
[FOR <condition>]
[DESCENDING]
[UNIQUE | DISTINCT | PRIMARY]

or to create dBASE III-compatible .NDX index files:

INDEX ON <key exp> TO <.ndx filename> [UNIQUE]

For DB and SQL tables:

INDEX ON <field list>
PRIMARY | TAG <tag name> [UNIQUE]

<key exp>

For DBF tables, <key exp> can be a dBL expression of up to 220 characters that includes field names, operators, or functions. The maximum length of the key—the result of the evaluated index <*key exp*>—is 100 characters.

<field list>

For Paradox and SQL tables, indexes can't include expressions; however, you can create indexes based on one or more fields. In that case, you specify the index key as a <field list>, separating the name of each field with a comma.

TAG <tag name>

Specifies the name of the index tag for the index

OF <.mdx filename>

Specifies the .MDX multiple index file that *dBASE Plus* adds new index tags to. If you do not specify an .MDX file, index tags are added to the production .MDX file. If you specify a file that doesn't exist, *dBASE Plus* creates it and adds the index tag name. By default, *dBASE Plus* assigns an .MDX extension and saves the file in the current directory.

TO <.ndx filename>

Specifies the name of an .NDX index file.

FOR <condition>

Restricts the records *dBASE Plus* includes in the index to those meeting the specified <condition>.

DESCENDING

Creates the index in descending order (Z to A, 9 to 1, later dates to earlier dates). Without DESCENDING, INDEX creates an index in ascending order.

UNIQUE

For DBF tables, prevents multiple records with the same <key exp> value from being included in the index; *dBASE Plus* includes in the index only the first record with that value. For DB and SQL tables, specifies creating a distinct index which prevents entry of duplicate index keys in a table.

DISTINCT

Prevents multiple records with the same <key exp> value from being included in the table; any such attempt causes a key violation error. Records marked as deleted are never included in a DISTINCT index. DISTINCT indexes may be created for DBF tables only.

PRIMARY

Specifies that the index is the primary key for the table. For DBF tables, the PRIMARY index is a distinct index that is designated as the primary index; it currently has no other special meaning. For DB and SQL tables, the primary key has a specific meaning. A table may have only one primary key.

Description

Use INDEX to organize data for rapid retrieval and ordered display. INDEX doesn't actually change the order of the records in a table but rather creates an index in which records are arranged in numeric, alphabetical, or date order based on the value of a key expression. Like the index of a book, with ordered entries and corresponding page numbers, an index file contains ordered key expressions with corresponding record numbers. When the table is used with an index, the contents of the table appear in the order specified by the index.

DBF expression indexes

To index on multiple fields in a DBF table, you must create an expression index. When combining fields with different data types, use conversion functions to convert all the fields to the same data type. Most multi-field expression indexes are character type; numeric and date fields are converted to strings using the STR( ) and DTOS( ) functions. When using the STR( ) function, be sure to specify the length of the resulting string so that it matches the numeric field.

Note

Do not use the DTOC( ) function to convert a date to a string. In many date formats, the day comes before the month, or the month comes before the day and year, resulting in records in the wrong order.

To concatenate the fields, use the + or - operators.

Warning!

Do not create an index where the length of the index key expression varies from record to record. Specifically, do not use TRIM( ) or LTRIM( ) to remove blanks from strings unless you compensate by adding enough spaces to make sure the index key values are all the same length. The - operator concatenates strings while rearranging trailing blanks. Varied key lengths may cause corrupted indexes.

If a function is used in a key expression, keep in mind that the index is ordered according to the function output. Thus, when you use search for a particular key, you must search for the key expression as it was generated. For example, INDEX ON SOUNDEX(Name) TO Names creates an index ordered by the values SOUNDEX( ) returns. When attempting to find data by the key value, you would have to use something like SEEK SOUNDEX("Jones") rather than SEEK "Jones".

FOR <condition> limits the records that are included in the index to those meeting the specified condition. For example, if you use INDEX ON Lastname + Firstname TO Salaried FOR Salary > 24000, *dBASE Plus* includes only records of employees with salaries higher than $24,000 in the index. The FOR condition can't include calculated fields.

The following built-in functions may be used in the index <key exp> and FOR <condition> expressions of a DBF index tag.

ABS( ) COS( ) FIELD( ) MEMLINES( ) RTOD( )

ACOS( )	CTOD( )	FLOOR( )	MIN( )	SECONDS( )
ANSI( )	CTODT( )	FV( )	MLINE( )	SIGN( )
ASC( )	CTOT( )	HTOI( )	MOD( )	SIN( )
ASIN( )	DATABASE( )	ID( )	MONTH( )	SOUNDEX( )
AT( )	DATE( )	INT( )	OEM( )	SPACE( )
ATAN( )	DAY( )	ISALPHA( )	OS( )	SQRT( )
ATN2( )	DBF( )	ISBLANK( )	PAYMENT( )	STR( )
BITAND( )	DELETED( )	ISLOWER( )	PI( )	STUFF( )
BITLSHIFT( )	DIFFERENCE( )	ISUPPER( )	PROPER( )	SUBSTR( )
BITNOT( )	DOW( )	ITOH( )	PV( )	TAN( )
BITOR( )	DTOC( )	LEFT( )	RAND( )	TIME( )
BITRSHIFT( )	DTOR( )	LEN( )	RAT( )	TRIM( )
BITSET( )	DTOS( )	LIKE( )	RECNO( )	TTIME( )
BITXOR( )	DTTOC( )	LOG( )	RECSIZE( )	TTOC( )
BITZRSHIFT( )	ELAPSED( )	LOG10( )	REPLICATE( )	UPPER( )
CEILING( )	EMPTY( )	LOWER( )	RIGHT( )	VAL( )
CENTER( )	EXP( )	LTRIM( )	ROUND( )	VERSION( )
CHR( )	FCOUNT( )	MAX( )	RTRIM( )	YEAR( )

Index sort order

In an index, records are usually arranged in ascending order, with lowest key values at the beginning of the index. Using the DOS Code Page 437 (U.S.) character set, character keys are ordered in ASCII order (from A to Z and then from a to z); numeric keys are ordered from lowest to highest numbers; and date keys are ordered from earliest to latest date (a blank date is higher than all other dates). Use the UPPER( ) function on the key expression to convert all lowercase letters to uppercase and achieve alphabetical order for character-type indexes.

Note

Most non-U.S. character sets provide a different sort order for characters than the DOS Code Page 437 character set.

You can reverse the order of an index, arranging records in descending order, by including the DESCENDING keyword. (You can use DESCENDING only when building .MDX tags.)

Distinct, primary, and unique indexes

You may use an index to ensure that there are no duplicate key values. For example, in a table of customers, each customer is assigned their own unique customer ID number. To prevent an existing customer ID number from being used by another customer, you can create a special kind of index on the customer ID field. For DB and SQL tables, this type of index is called a unique index; the key value for each record in the table must be unique. For DBF tables, this type of index is called a distinct index; a unique index for a DBF table has a different meaning. For clarity, the DBF terms are used.

A distinct index is created with the DISTINCT option for DBF tables, and the UNIQUE option for DB and SQL tables. When a table has a distinct index, any attempt to create a duplicate key entry, either by adding a new record with a duplicate value or by changing an existing record so that its key field(s) duplicates another record, causes a key violation error. The new or changed record is not written to the table. Distinct indexes for DBF tables never include records that are marked as deleted.

A table may also have one distinct index designated as its primary index, or primary key. A primary index is usually created for the ID field or fields that uniquely identify each record in the table. For example, while you may index on the customer's name, their ID field is what uniquely identifies each customer, and that is the field you use for the primary key. For DB tables, a

table's primary key determines the default order for the records in the table, and you must have a primary key to create other secondary indexes. For DBF tables, a primary key currently has no special meaning, other than self-documenting the primary key field(s) of the table. The PRIMARY clause is used to create the primary index. For DB and SQL tables, a primary index may have no other options other than the field list.

DBF tables support a kind of index that allows duplicate key values in the table, but only shows the first such record in the index. These are called unique indexes, not to be confused with the distinct unique indexes used by DB and SQL tables. For example, you may be interested in the names of the cities in which your customers reside. By using a unique index, each city is listed once (alphabetically), no matter how many customers you have in that city.

A record's index key value is tested for uniqueness only when the record is added or updated. For example, suppose you have a unique index on the City field, and have records in both "Bismark" and "Fargo". If you append another record in "Bismark", it does not appear in the index, although the table is updated with the new record. If you then change the first record, which was listed in the index, from "Bismark" to "Fargo", then it too becomes hidden because there is already a "Fargo" in the index. It also does not automatically expose the other record with "Bismark", because that record was not updated; no records in "Bismark" are in the index at that moment. REINDEX explicitly updates all key values in a unique index.

Indexing a table with SET UNIQUE ON has the same effect as INDEX with the UNIQUE option. With DB and SQL tables, it creates a distinct index. With DBF tables, it creates a unique index.

Using indexes

Once a table has been indexed, use LOOKUP( ), SEEK, and SEEK( ) to retrieve data. The structure of an index file allows these commands to quickly locate values of the key expression.

Whenever data in key fields is modified, *dBASE Plus* automatically updates all open index files. Index files closed when changes are made in a table can be opened and then updated using REINDEX.

Multiple index files simplify updating indexes, since *dBASE Plus* updates all indexes with tag names listed in .MDX files specified with USE...ORDER or SET ORDER. *dBASE Plus* automatically opens a production .MDX file, if one exists, when you open the associated table.

INDEX...TAG creates an index and adds the tag name to a multiple index file. If you don't include OF <filename>, INDEX...TAG adds the tag name to the production .MDX file. *dBASE Plus* creates the production .MDX or the specified file if it doesn't already exist.

INDEX is similar to SORT, another command that allows ordering of a table. Unlike INDEX, though, SORT physically rearranges the table records, a time-consuming process for large files. To maintain the sorted order, either new records must be inserted in their proper position, which is also very time-consuming, or the entire table must be resorted. Also, SORT doesn't support LOOKUP( ), SEEK, or SEEK( ), making the process of locating data in a sorted table much slower.

At the end of an indexing operation, the new index file is the master index, and the record pointer is positioned at the first record of the new indexed.

OODML

Use the Database object's *createIndex( )* method.

ISBLANK( )

Topic group Related topics

Determines if a specified field or expression is blank.

Syntax

ISBLANK(<exp>)

<exp>

An expression of any data type.

Description

ISBLANK( ) returns *true* if a specified expression is blank or *null*; *false* if it contains data. A field is blank if it has never contained a value or if you used the BLANK command on it. ISBLANK( ) returns a different result from EMPTY( ) when used on numeric fields; ISBLANK( ) differentiates between zero and blank values, while EMPTY( ) does not.

ISBLANK( ) is especially useful when performing functions such as averaging, since it ensures that blank values are not included in the calculation. If you don't need to differentiate between 0 or blank values in numeric fields, you can use either ISBLANK( ) or EMPTY( ).

OODML

No equivalent.

ISTABLE( )

Topic group Related topics

Tests for the existence of a table in a specified database.

Syntax

ISTABLE(<filename>)

<filename>

The name of the table to search for. You cannnot use a filename skeleton for ISTABLE( ). If you do, it will return *false*.

Description

Use ISTABLE( ) to confirm the existence of a table. If the table exists, ISTABLE( ) returns *true*; otherwise it returns *false*.

OODML

Use the Database object's *tableExists( )* method.

KEY( )

Topic group Related topics Example

Returns the key expression of the specified index.

Syntax

KEY([<.mdx filename>,] [<index position expN> [,<alias>]])

<.mdx filename expC>

The .MDX file that contains the index tag you want to check. The .MDX must be opened in the specified work area. If omitted, all open indexes, including the production .MDX file, are searched.

<index position expN>

the numeric position of the index tag in the specified .MDX file, or the position of the index in the list of open indexes.

<alias>

The work area you want to check.

Note

Unlike most functions, the first parameter is optional. If you omit the first parameter, the remaining parameters shift forward one; the second parameter becomes the first parameter, and so on.

Description

KEY( ) returns a string containing the key expression of the specified index. To see the value of the key expression for a given record, store the string returned by KEY( ) in a private variable. Then use macro substitution to evaluate the expression.

The index must be referenced by number. If you do not specify an .MDX file to check, index numbering in the list of open indexes is complicated if you have open .NDX indexes or you have open non-production .MDX files. For more information on index numbering, see SET INDEX. Either way, it is often easier to reference an index tag by name by using the TAGNO( ) function to get the corresponding position number.

If you do not specify an index tag, KEY( ) checks the current master index tag and returns an empty string if there is no master index.

If the specified .MDX file or index tag does not exist, KEY( ) returns an empty string.

OODML

No equivalent.

KEYMATCH( )

Topic group Related topics

Indicates if a specified expression is found in an index.

Syntax

KEYMATCH (<exp> [,<index number> [,<alias>]])

where <index number> is:

<index position expN> | [<.mdx filename expC>,] <tag expN>

<exp list>

The expression, of the same data type as the index, that you want to look for.

<index position expN>

The numeric position of the index in the list of open indexes for the table.

<.mdx filename expC>

The .MDX file that contains the index tag you want to check. The .MDX must be opened in the specified work area.

<tag expN>

The numeric position of the index tag in the specified .MDX file.

<alias>

The work area you want to check.

Description

The KEYMATCH( ) function determines if a specified key expression is found in a particular index. KEYMATCH( ) returns *true* or *false* to indicate whether the specified expression was found. SET EXACT controls whether exact matches of character string data is required.

A primary use of the KEYMATCH( ) function is to check for duplicate values when adding records. Unlike SEEK( ), KEYMATCH( ) looks for a matching index value without moving the record pointer and disturbing the current state of the record buffer.

KEYMATCH( ) ignores the settings for SET FILTER and SET KEY TO, ensuring the integrity of data in a table even when you work with a subset of the table records. KEYMATCH( ) honors SET DELETED, so that when SET DELETED is ON, existing key values in records marked as deleted are ignored, as if those records did not exist.

If you specify only an expression (<exp>) whose value you want to match, KEYMATCH( ) searches the current master index for an index key with the same value.

To search indexes other than the current master index, you must specify the index by index position. There are two ways to do this:

1. By the index's position in the list of open indexes. Index numbering is complicated if you have open .NDX indexes or open non-production .MDX files. For information on index numbering, see SET INDEX.
    - By an index tag's position in an .MDX file. If you do not specify <.mdx filename expC>, the production .MDX is used.

Either way, it is often easier to reference an index tag by name by using the TAGNO( ) function to get the corresponding position number.

OODML

No equivalent.

LIST

Topic group Related topics

Displays records from the current table in the result pane of the Command window.

Syntax

LIST
[<*scope*>]
[FOR <*condition 1*>]
[WHILE <*condition 2*>]
[[FIELDS] <*exp list*>]
[OFF]
[TO FILE <*filename*>]
[TO PRINTER]

Description

Both LIST and DISPLAY display records in the results pane of the Command window. There are two differences between the commands:

LIST displays continuously until complete, while DISPLAY pauses after each screenful of information.
The default scope of LIST is ALL, while the default scope of DISPLAY is NEXT 1, the current record only.

Because LIST does not pause between screens, it is more appropriate when directing output to a file or printer. For more information on the options of LIST, see DISPLAY.

OODML

No equivalent.

LKSYS( )

Topic group Related topics Example

Returns information about a locked record or file.

Syntax

LKSYS(<expN>)

<expN>

A number representing the information for LKSYS( ) to return:

Value	Returns
0	Time when lock was placed
1	Date when lock was placed
2	Login name of user who locked record or file
3	Time of last update or lock
4	Date of last update or lock
5	Login name of user who last updated or locked record or file

Description

LKSYS( ) returns multiuser information contained in a _DBASELOCK field of a DBF table. For LKSYS( ) to return information, the current table must have a _DBASELOCK field. Use CONVERT to add a _DBASELOCK field to a table. If the current table doesn't contain a _DBASELOCK field, LKSYS( ) returns an empty string for any value of <expN>.

Note

LKSYS( ) works only with DBF tables.

LKSYS( ) always returns a string. When LKSYS( ) returns a date, it is a string containing the date in the current date format dictated by SET DATE and SET CENTURY. Use CTOD( ) to convert the date string to a date.

When a record is locked, either explicitly or automatically, the time, date, and login name of the user placing the lock are stored in the _DBASELOCK field of that record. When a file is locked, this same information is stored in the _DBASELOCK field of the first physical record in the table.

Passing 0, 1, or 2 as arguments to LKSYS( ) returns values only after an attempted file or record lock has failed. If a file or record lock on a converted table fails, the information for LKSYS( ) arguments 0, 1, and 2 is written to a buffer from the record's _DBASELOCK field. If you then pass 0, 1, or 2 to LKSYS( ), the information is read from the buffer. The buffer isn't overwritten until you attempt another lock that fails. Thus, 0, 1, and 2 always return the information that was current at the time of the last lock failure.

You can pass 3, 4, or 5 as arguments to LKSYS( ) whether or not the current record or file is currently locked. These arguments return information about the last successful record or file lock. When you pass any of these arguments to LKSYS( ), it returns information directly from the _DBASELOCK field rather than from an internal buffer.

If you pass 2 or 5 to obtain a user login name, and the _DBASELOCK field is only 8 characters wide, LKSYS( ) returns an empty string. The first 8 characters of a _DBASELOCK field are the count, time, and date information of the last update or lock, so the field must be wider than 8 characters to fit part or all of the login user name. Set the width of the field with CONVERT.

OODML

Check the properties of the rowset.fields["_DBASELOCK"] field.

LOCATE

Topic group Related topics Example

Searches a table for the first record that matches a specified condition.

Syntax

LOCATE
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]

<scope>
FOR <condition 1>
WHILE <condition 2>

The scope of the command. The default scope is ALL. LOCATE is usually used with a FOR condition.

Description

LOCATE performs a sequential search of a table and tests each record for a match to the specified condition. If a match is found the record pointer is left at that record. Issuing CONTINUE resumes the search, allowing additional records meeting the specified condition to be found.

Whenever a match is found, FOUND( ) returns *true*. If match is not found, the record pointer is left after the last record checked, which usually leaves it at the end-of-file. Also, FOUND( ) returns *false*.

If SET TALK is ON, LOCATE will display the record number of the matching record in the result pane of the Command window if you are searching a DBF table. If no match is found, LOCATE will display "End of Locate scope"

Because the default scope of the command is ALL, issuing LOCATE with no options will move the record pointer to the first record in the table, because that is the first matching record in that scope. However, there is no practical reason to use LOCATE in this manner. A FOR condition is usually used with LOCATE to find records that match a condition. An understanding of command scope is essesntial to using LOCATE effectively.

LOCATE does not require an indexed table; however, if an index is in use, LOCATE follows its index order. When using the = operator to compare strings, LOCATE uses the rules established by SET EXACT to determine whether the strings match. Use the == operator to perform exact matches regardless of SET EXACT.

The search commands LOCATE and SEEK are each designed for use under particular conditions. LOCATE is the most flexible, accepting expressions of any data type as input and searching any field of a table. For large tables, however, a sequential search using LOCATE might be slow.

Use SEEK or SEEK( ) for greater speed. Both conduct an indexed search, similar to looking up a topic in a book index and turning directly to the appropriate page, allowing information to be found almost immediately. Once you use the INDEX command to create an index for a table, SEEK uses this index to quickly identify an appropriate record.

You can use SEEK and LOCATE in combination. Use SEEK to quickly narrow down a search and then use LOCATE with the appropriate scope to find the exact you're looking for.

OODML

Use the Rowset object's *beginLocate( )* and *applyLocate( )* methods.

LOCK( )

Topic group Related topics

Locks the current record or a specified list of records in a table.

Syntax

LOCK([<*record list expC*>,<*alias*>] | [<alias>])

<list expC>

The list of record numbers to lock, separated by commas.

<alias>

The work area in which to lock records.

Description

LOCK( ) is identical to RLOCK( ). For more information, see RLOCK( ).

LOOKUP( )

Topic group Related topics Example

Searches a field for a specified expression and, if the expression is found, returns the value of a field within the same record.

Syntax

LOOKUP(<return field>, <exp>, <lookup field>)

<return field>

The field whose value you want to return if a match is found.

<exp>

The expression to look for in the <lookup field>. Specify an alias when referring to fields outside the current work area.

<lookup field>

The field you want to search for the value <exp>.

The <return field> and <lookup field> are usually fields in the same table, a table that is not in the current work area. Use the alias name and alias operator (->) to reference fields in other tables.

Description

LOOKUP( ) looks for the first record where <lookup field> matches the specified expression <exp>. The record pointer is left at the matching record. If no match is found, the record pointer is left at the end-of-file. Either way, LOOKUP( ) then returns the value of <return field>.

Therefore, if no match is found, LOOKUP( ) returns the blank value for that field, either an empty string (""), zero, a blank date, or *false*, depending on the data type of <lookup field>. Calling FOUND( ) will also return *true* or *false* to indicate if the search was successful.

LOOKUP( ) performs a sequential search, unless an index whose key matches <lookup field> is available in the lookup table. To minimize the time LOOKUP( ) takes to search a table, you should create index keys for your most common lookups.

Because LOOKUP( ) moves the record pointer you can perform a lookup with related tables, where the <lookup field> is in the parent table, and <return field> is in the child table.

OODML

No equivalent.

LUPDATE( )

Topic group Related topics

Returns the date of the last change to a table.

Syntax

LUPDATE([<alias>])

<alias>

The work area you want to check.

Description

LUPDATE( ) returns the last update date of the specified table. If no table is open, LUPDATE( ) returns a blank date.

OODML

No equivalent. You may use functions to check the last update date of the table file.

MDX( )

Topic group Related topics Example

Returns the names of a DBF table's open .MDX index files.

Syntax

MDX([<mdx expN>[, <alias>]])

<mdx expN>

A number indicating which open .MDX file whose name to return.

<alias>

The work area you want to check.

Description

MDX( ) returns the name of an .MDX file open in the current or specified work area. .MDX files are numbered in the order in which they were opened. The production .MDX file, the one with the same name as the DBF file, is number 1.

If <mdx expN> is omitted, the name of the .MDX file containing the current master index tag is returned.

MDX( ) includes the drive letter (and colon) in the filename. If SET FULLPATH is ON, MDX( ) also returns the directory location of the .MDX file in addition to the drive and name.

If <mdx expN> is higher than the number of open .MDX files, or if you do not specify an index order number and the master index is an .NDX file, MDX( ) returns an empty string (""). MDX( ) also returns an empty string if there is no .MDX file open.

OODML

No equivalent.

MEMLINES( )

Topic group Related topics

Returns the number of lines in a memo field.

Syntax

MEMLINES(<memo field> [,<line length expN>])

<memo field>

The memo field the MEMLINES( ) function operates on.

<line length expN>

Specifies the line length used in calculating the number of lines in a memo field. <expN> can be set to any number from 8 to 255. If <expN> is not specified, MEMLINES( ) calculates each line using the memo width specified using the SET MEMOWIDTH command.

Description

The MEMLINES( ) function returns the number of lines in a memo field based on the memo width specified by the line length parameter. If you don't specify a line length, MEMLINES( ) uses the width specified by SET MEMOWIDTH, which defaults to 50.

If a word doesn't completely fit within the remainder of a line, MEMLINES( ) wraps that word and everything on the line following it to the beginning of the next line. If the number of characters in a word is longer than the default or specified memo field line length, MEMLINES( ) truncates the word at the end of the line and includes the remainder of the word at the beginning of the next line.

A carriage return/line feed combination in the memo text always starts a new line. Note that if the carriage return/line feed is at the end of the memo field contents, the empty blank line that follows it is counted in the total line count.

OODML

No equivalent. You cannot accurately determine the amount of text that can fit on a line when using proportional fonts.

MLINE( )

Topic group Related topics

Extracts a specified line of text from a memo field in the current record.

Syntax

MLINE(<memo field> [, <line number expN > [, <line length expN>]])

<memo field>

The memo field the MLINE( ) function operates on.

<line number expN >

The number of the line in the memo field returned by the MLINE( ) function. The default for <line number expN> is 1, the first line.

<line length expN >

The number that determines the length of a line in the memo field. <line length expN> can be set to any number from 8 to 255. If <line length expN> is not set, the SET MEMOWIDTH setting specifies the length of the line.

Description

MLINE( ) returns a specified line of text from a memo field. MLINE( ) treats the text of the memo field as if it were wordwrapped within a display width specified by the SET MEMOWIDTH setting or by <line length expN>.

If a word doesn't completely fit within the remainder of a line, MLINE( ) wraps that word and everything on the line following it to the beginning of the next line. If the number of characters in a word is longer than the default or specified memo field line length, MLINE( ) truncates the word at the end of the line and includes the remainder of the word at the beginning of the next line.

OODML

No equivalent.

NDX( )

Topic group Related topics

Returns the names of a DBF table's open .NDX files.

Syntax

NDX([<ndx expN> [, <alias>]])

<ndx expN>

A number indicating which open .NDX file whose name to return.

<alias>

The work area you want to check.

Description

NDX( ) returns the name of the .NDX file open in the current or specified work area. .NDX files are numbered in the order in which they were opened. The first one is number 1.

If <ndx expN> is omitted, the name of the .NDX file containing the current master index tag is returned.

NDX( ) includes the drive letter (and colon) in the filename. If SET FULLPATH is ON, NDX( ) also returns the directory location of the .NDX file in addition to the drive and name.

If <ndx expN> is higher than the number of open .NDX files, or if you do not specify an index order number and the master index is an index tag in an .MDX file, NDX( ) returns an empty string ("").

OODML

No equivalent.

OPEN DATABASE

Topic group Related topics

Establishes a connection to a database server or a database defined for a specific directory location.

Syntax

OPEN DATABASE <database name>
[LOGIN <username>/<password>]
[WITH <option string>]
[AUTOEXTERN]

<database name>

The name, or alias, of the database you want to open. Database aliases are created using the BDE Administrator.

<user name>/<password>

The user name and password, separated by a slash, required to access the database.

WITH <option string>

Character string specifying server-specific information required to establish a database server connection. For information about establishing database server connections, refer to your Borland SQL Link documentation, and contact your network or database administrator for specific connection information.

AUTOEXTERN

Treat all stored procedures as EXTERN. This eliminates the need for the user to EXTERN SQL any stored procedure calls. Once the database is open, the stored procedures are immediately available. For use with Interbase and Oracle databases only.

Description

The OPEN DATABASE command is used to establish a connection with a database defined with the BDE Administrator. When opening a database, you need to specify whatever login parameters and database-specific information that connection requires. Typically, your network or system administrator can provide you with the information necessary to establish connections to established databases and database servers at your site.

OODML

Use a Database object.

ORDER( )

Topic group Related topics Example

Returns the name of the current master index.

Syntax

ORDER([<alias>])

<alias>

The work area you want to check.

Description

ORDER( ) returns the name of the current master index. For DBF tables, this could be either the name of an index tag in an .MDX file, or the name of an .NDX file (the name only, no drive or extension as returned by the NDX( ) function). For all other table types, the name is the name of an index tag.

ORDER( ) returns an empty string ("") if the table is in its natural order: either its primary key order, if it has a primary key; or no active index.

Some routines need to use a specific index. Use ORDER( ) to get the name of the current master index before switching to the desired index and then use the SET ORDER command to later restore the master index.

OODML

Check the *indexName* property of the Rowset object.

PACK

Topic group Related topics

Removes all records from a .DBF table that have been marked as deleted. Adding an autoincrement field will automatically pack a .DBF table.

Syntax

PACK

Description

Use PACK to remove records from the current .DBF table that were previously marked as deleted by the DELETE command. You must open the table for exclusive use before using PACK.

After you execute a PACK command, the disk space used by the deleted records is reclaimed when the table is closed. All open index files are automatically re-indexed after PACK is executed. (Use REINDEX to update closed indexes.)

Use PACK with caution. Records that have been marked for deletion but not yet eliminated with PACK can be undeleted and restored to a table using RECALL. Records eliminated with PACK are permanently lost and can't be recovered.

SET DELETED ON provides many of the benefits of PACK without actually removing records from a table. With SET DELETED ON, most commands function as if records marked for deletion had been eliminated from a table.

Because PACK requires the exclusive use of a table, it may be difficult to find a time to PACK a table for applications that run continuously. Also for large tables, PACK is time-consuming and requires enough disk space to make a copy of the table. Consider recycling deleted records instead, which is quicker and safer. For an example of how to implement record recycling, see the examples for APPEND and BLANK.

To permanently remove all records of the current table in one step, use the ZAP command.

OODML

Use the Database object's *packTable( )* method.

RECALL

Topic group Related topics Example

Restores records that were previously marked as deleted in the current DBF table.

Syntax

RECALL
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]

<scope>
FOR <condition 1>
WHILE <condition 2>

The scope of the command. The default scope is NEXT 1, the current record only.

Description

Use RECALL to undelete records that have been marked as deleted in the current DBF table with DELETE but have not yet been removed with PACK. Executing DELETE marks the record as deleted but doesn't physically remove them from the table. If SET DELETED is ON (the default), these deleted records cannot be seen. RECALL reverses this process, unmarking the records and fully restoring them to the table.

Records eliminated with PACK or ZAP are permanently removed and can't be recovered using RECALL.

When using RECALL, SET DELETED should be OFF; otherwise you will not be able to see the deleted records you want to recall. Using RECALL on records that are not marked as deleted has no effect.

OODML

Soft deletes are not supported.

RECCOUNT( )

Topic group Related topics

Returns the number of records in a table.

Syntax

RECCOUNT([<alias>])

<alias>

The work area you want to check.

Description

RECCOUNT( ) retrieves a count of a table's records from the table header, which holds information about the table structure. In contrast, COUNT with no options yields a record count by actually counting the table's records using the table's current filter, key constraints, the

setting of SET DELETED and so on. RECCOUNT( ) includes all records, even those marked as deleted, and is always instantaneous; COUNT is not.

If no table is active in the specified work area, RECCOUNT( ) returns zero.

You can use RECSIZE( ) in combination with RECCOUNT( ) to determine the approximate size, in bytes, of a table.

OODML

In some cases, the Rowset object's *rowCount( )* method will return the same value.

RECNO( )

Topic group Related topics

For DBF tables, returns the current record number. For all other table types, returns a bookmark of the current position in a table.

Syntax

RECNO([<alias>])

<alias>

The work area you want to check.

Description

RECNO( ) returns the current record number of the table in the current or a specified work area, if that table is a DBF table. For all other table types, RECNO( ) behaves like BOOKMARK( ), returning a bookmark. If no table is open in the specified work area, RECNO( ) returns a value of 0.

If the record pointer is at end-of-file (past the last record in the table), RECNO( ) returns a value that is one more than the total number of records in the table. Therefore, RECNO( ) returns a value of 1 if there are no records in the table—RECCOUNT( ) would return zero.

The use of BOOKMARK( ) is recommended instead of RECNO( ). Besides returning a consistent data type with all tables, with BOOKMARK( ) you can bookmark the position at the end-of-file and GO back to it. Although RECNO( ) will return a record number for the end-of-file, you cannot GO to that record number, because there actually is no record with that number.

OODML

Use the Rowset object's *bookmark( )* method.

RECSIZE( )

Topic group Related topics Example

Returns the number of bytes in a record of a table.

Syntax

RECSIZE([<alias>])

<alias>

The work area you want to check.

Description

RECSIZE( ) returns the number of bytes in a record of a table in the current or specified work area. If no table is open in the specified work area, RECSIZE( ) returns a value of zero.

LIST STRUCTURE and DISPLAY STRUCTURE also show the size of a table's records.

OODML

Use a loop to add up the *length* properties of the Field objects in the *fields* array.

REFRESH

Topic group Related topics

Updates data buffers to reflect the latest changes to data.

Syntax

REFRESH [<alias>]

<alias>

The work area to refresh.

Description

Use REFRESH to update specified work area data buffers so that data you display reflects the latest changes made to tables by other users.

OODML

Use the Rowset object's *refresh( )* method or the Query object's *requery( )* method.

REINDEX

Topic group Related topics Example

Regenerates all open index files in the current work area.

Syntax

REINDEX

Description

Use REINDEX to manually regenerate all open indexes in the current work area. In a normal application, indexes remain open as long as their tables are open. These indexes are automatically updated whenever there is a change to the table, so there is no need to manually REINDEX.

You would use REINDEX if your application uses non-production .MDX files or .NDX files that are not always open. To update these indexes, open them with the corresponding table and issue REINDEX.

You might also use REINDEX if you suspect that the index files have been damaged. REINDEX rebuilds the entire index file from scratch.

You must have exclusive use of a table to REINDEX it.

OODML

Use the Database object's *reindex( )* method.

RELATION( )

Topic group Related topics

Returns the link expression defined with the SET RELATION command.

Syntax

RELATION(<expN> [,<alias>])

<expN>

The number of a relation that you want to return.

<alias>

The work area you want to check.

Description

RELATION( ) returns a string containing the expression that links one table with another that was defined with the SET RELATION command. You must specify the number of the relation; if the table in the current or specified work area is linked to only one table, that <expN> is the number 1. RELATION( ) returns an empty string ("") if no relation is set in the <expN> position.

Use RELATION( ) to save the link expressions of all SET RELATION settings for later use when restoring relations. To save the target table (the table into which you SET a RELATION), use the TARGET( ) function.

OODML

Check the detail Rowset object's *masterFields* and *masterRowset* properties, or the detail Query object's *masterSource* property to determine the nature of the master-detail linkage.

RELEASE AUTOMEM

Topic group Related topics

Clears automem variables from memory.

Syntax

RELEASE AUTOMEM

Description

Automem variables are private or public memory variables that have the same name as the fields of the currently selected table. These variables can be created manually, or with the STORE AUTOMEM, CLEAR AUTOMEM, or USE...AUTOMEM commands.

RELEASE AUTOMEM releases any private or public memory variables that have the same name as one of the fields in the currently selected table, no matter how or for what purpose the variable was created.

Closing a table or moving to another work area doesn't automatically release a table's associated automem variables. *dBASE Plus* doesn't recognize a variable as an automem variable, even if it was created as an automem variable, if it doesn't have the same name as a field in the current table. But because automem variables are usually private variables, and private variables are automatically released when the routine that created them is complete, there is rarely any reason to issue RELEASE AUTOMEM in an application.

OODML

The Rowset object contains an array of Field objects accessed through its *fields* property. These Field objects have *value* properties that may be programmed like variables.

RENAME TABLE

Topic group Related topics

Changes the name of a specified table.

Syntax

RENAME TABLE <old table name> TO <new table name>
[[TYPE] PARADOX | DBASE]

<old table name>

The table you want to rename.

<new table name>

The new name of the table. If you rename a table in a database, you must specify the same database as the destination of the new table. Also, the new table name must be the same type as the old table.

[TYPE] PARADOX | DBASE

Specifies the type of table you want to rename, in case you do not specify a file extension with <old table name>. This option overrides the current SET DBTYPE setting. The TYPE keyword is included for readability only; it has no effect on the operation of the command.

Description

Use the RENAME TABLE command to change the name of a table and its associated files, if any. You cannot rename an open table, and the new table name cannot already exist in the same directory or database.

OODML

Use the Database object's *renameTable( )* method.

REPLACE

Topic group Related topics

Replaces the contents of fields with data from expressions.

Syntax

REPLACE
<field 1> WITH <exp 1> [ADDITIVE]
[, <field 2> WITH <exp 2> [ADDITIVE]...]
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]
[REINDEX]

<field> WITH <exp>

Designates fields to be replaced by the value of the specified expressions. Multiple fields of a record may be changed by including additional <field n> WITH <exp n> expressions, separated by commas.

ADDITIVE

Adds text to the end of memo field text instead of replacing existing text. You can use ADDITIVE only when the specified field is a memo field in a DBF table.

<scope>
FOR <condition 1>
WHILE <condition 2>

The scope of the command. The default scope is NEXT 1, the current record only.

REINDEX

Specifies that all affected indexes are rebuilt once the REPLACE operation finishes. Without REINDEX, *dBASE Plus* updates all open indexes after replacing each record in the scope. When replacing many or all records in a table that has multiple open indexes, REPLACE executes faster with the REINDEX option.

Description

The REPLACE command overwrites a specified field with new data. The field you select can be any type, including memo fields. (To replace binary or OLE fields, use REPLACE BINARY and REPLACE OLE.) The field and the expression specified by the WITH clause must have the same data type.

When storing a long string or the contents of a memo field into a shorter character field, the data is truncated. Use the ADDITIVE option to add a character string to the end of existing memo field text. You can leave a blank space at the beginning of the string to provide proper spacing.

Be careful when replacing data in the key fields of the current master index, in more than one record (that is, with the <scope>, WHILE, or FOR options).*dBASE Plus* automatically updates all open index files after a REPLACE operation finishes. After replacing data that changes the value in the key field in the master index, the record and the record pointer immediately move to the position in the index based on the new value of a key. If replacement in the key field causes a record (and the pointer) to move down past other records that fall within the scope or meet the specified conditions, those records are not replaced. If replacement in the key field causes a record to move up before records that have already been replaced, those records may be replaced again.

To make replacements to an indexed table's key field, you may place the table in natural order with the SET ORDER TO command, or use other techniques, one of which is shown in the example. Replacements in key fields other than the key fields of the master index don't affect the order of the current index and can be made over multiple records without complications.

When replacing a Numeric or Float field in a DBF table, the magnitude of the new value should not exceed the integer portion of the field. For example, if the Numeric field is defined as width 4 with 1 decimal place, you cannot have a number greater than 99.9. If so, the field contents are replaced with an approximation to the new value in scientific notation, if it will fit; otherwise the field contents are replaced with asterisks, destroying stored data. Scientific notation requires a field width of at least 7 characters. This condition is not an error, but *dBASE Plus* will display a numeric overflow warning message in the result pane of the Command window.

Other field types that store numbers, including Long and Short integers, have a numeric range they support. Make sure that the number you attempt to store does not exceed those ranges.

Use the alias operator in the <field> (that is, alias–>field) to REPLACE fields in tables other than the currently selected table. You may mix fields from different tables in the same REPLACE statement, although the scope of the command is based on the current table. If there is no relation between the current table and other tables, traversing the current table—for example, because of an ALL scope—does not move the record pointer in the other tables.

OODML

Assign values directly to the *value* properties of the Field objects in the Rowset object's *fields* array (in a loop that traverses the rowset if necessary).

REPLACE AUTOMEM

Topic group Related topics

Replaces fields in the current table that have corresponding automem variables.

Syntax

REPLACE AUTOMEM

Description

Automem variables are private or public memory variables that have the same name as the corresponding fields of the current table. Automem variables are used to hold data that will be stored in the fields of records. You can manipulate data stored in automem variables as memory variables rather than as field values, and you can validate the data before storing the data to a table.

Create a set of automem variables for the fields in a table with USE...AUTOMEM, CLEAR AUTOMEM, or STORE AUTOMEM (or create the variable manually). To add new records to a table and fill the fields with values from corresponding automem variables, use APPEND AUTOMEM. To update the fields of existing records with values from corresponding automem variables, use REPLACE AUTOMEM.

Use REPLACE AUTOMEM to update all the fields of a record without having to name the fields. By contrast, with the REPLACE command, you need to name every field you want updated.

Remember that an automem variable and its corresponding field have the same name. When a command allows an argument that could be either a field or a private or public memory variable, *dBASE Plus* assumes the argument refers to a field. To distinguish the memory variable from the field, prefix the names of automem variables with M->.

REPLACE AUTOMEM updates the current record. It can't update all records within a specified scope or all records matching a condition, as the REPLACE command can with the options <scope>, FOR <condition>, and WHILE <condition>.

REPLACE AUTOMEM doesn't replace field data with data from a memory variable with the same name but of a different data type. Those variables are ignored. If a field does not have a corresponding private or public variable with the same name, that field is left unchanged.

Note: Read-only field type - Autoincrement

Because APPEND AUTOMEM and REPLACE AUTOMEM write values to your table, the contents of the read-only field type, Autoincrement, must be released before using either of these commands. In the following example, the autoincrement field is represented by "myAutoInc":

```
use table1 in 1
use table2 in 2
select 1      // navigate to record
store automem
release m->myAutoInc
select 2       // navigate to record
replace automem
```

OODML

The Rowset object contains an array of Field objects, accessed through its *fields* property. These Field objects have *value* properties that may be programmed like variables.

REPLACE BINARY

Topic group Related topics

Replaces the contents of a binary field with the contents of a binary file.

Syntax

REPLACE BINARY <binary field name> FROM <filename>
[TYPE <binary type user number>]

<binary field name>

The binary field of the current table that is replaced by the contents of <filename>.

FROM <filename>

Specifies the file to copy to the binary field in the current record. If you specify a file without including its extension, *dBASE Plus* assumes a .BMP extension; however, the file may be any type.

TYPE <binary type user number>

Specifies a number that can be used to identify the type of binary data being stored. By default, *dBASE Plus* attempts to detect the type of file and assigns the appropriate binary type. Use the BINTYPE( ) function to retrieve the type number. The range is from 1 to 32K – 1 for user-defined file types and 32K to 64K – 1 for predefined types (although any number may be specified within the allowable range).

Predefined binary type numbers	Description
1 to 32K – 1 (32,767)	User-defined file types
32K (32,768)	.WAV files
32K + 1 (32,769)	Image files

Description

Use REPLACE BINARY to copy a binary file to the current record's binary field. You can copy one binary file to each binary field of each record in a table.

While memo fields may contain types of information other than text, binary fields are recommended for storing images, sound, or any other binary or BLOB type data.

See class Image for a list of image types that *dBASE Plus* can automatically detect and display.

OODML

Use the Field object's *replaceFromFile( )* method. The binary type option is not supported.

REPLACE FROM ARRAY

Topic group Related topics

Transfers data stored in an array to the fields of the current table.

Syntax

REPLACE FROM ARRAY <array>
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]
[FIELDS <field list>]
[REINDEX]

<array>

The name of the array that you want to transfer data from.

<scope>
FOR <condition 1>
WHILE <condition 2>

The scope of the command. The default scope is REST. The dimensions and size of <array> also controls which records are updated.

FIELDS <field list>

Restricts data replacement to the fields specified by <field list>.

REINDEX

Specifies that all affected indexes are rebuilt once the REPLACE FROM ARRAY operation finishes. Without REINDEX, *dBASE Plus* updates all open indexes after replacing each record in the scope. When replacing many or all records in a table that has multiple open indexes, REPLACE FROM ARRAY executes faster with the REINDEX option.

Description

Use REPLACE FROM ARRAY to transfer values from an array into fields of the current table. REPLACE FROM ARRAY treats the columns in a one-dimensional array like a single record of fields; and treats a two-dimensional array like a table, with the rows (the first dimension) of the array like records, and the columns (the second dimension) like fields.

For a one-dimensional array, REPLACE FROM ARRAY will replace the first record in the command scope that matches the specified condition. If there is no specified scope and condition, the current record is replaced.

For a two-dimensional array, REPLACE FROM ARRAY will attempt to copy each row in the array to a record in the command scope that matches the specified condition until the end-of-scope or all rows of the array have been copied, in which case the record pointer is left at the last record replaced. As with the REPLACE command, be careful if are changing the values of the key fields of the current master index.

If the array has more columns than the table has fields, the additional elements are ignored. Similarly, if a two-dimensional array has more rows than can be copied to the table, the additional rows are ignored.

REPLACE FROM ARRAY does not replace memo (or binary) fields; these fields should not be counted when sizing the array, and cannot be included in the <field list>.

The data types of the array must match those of corresponding fields in the table you are replacing. If the data type of an array element and a corresponding field don't match, an error occurs.

OODML

Use a loop to copy the elements of the array into the *value* properties of the Field objects in the rowset's *fields* array, nested in another loop to traverse the rowset if necessary.

REPLACE MEMO

Topic group Related topics

Copies a text file into a memo field.

Syntax

REPLACE MEMO <memo field> FROM <filename>
[ADDITIVE]

<memo field>

The memo field to replace.

FROM <file name>

The name of the text file. The default extension is .TXT.

ADDITIVE

Causes the new text to be appended to existing text. REPLACE MEMO without the ADDITIVE option causes *dBASE Plus* to overwrite any text currently in the memo field.

Description

Use the REPLACE MEMO command to insert the contents of a text file into a memo field. You may use an alias name and the alias operator (that is, alias->memofield) to specify a memo field in the current record of any open table.

REPLACE MEMO is identical to APPEND MEMO, except that REPLACE MEMO defaults to overwriting the current contents of the memo field, and has the option of appending, while APPEND MEMO is the opposite.

While memo fields may contain types of information other than text, binary fields are recommended for storing images, sound, and other user-defined binary type information. Use OLE fields for linking to OLE documents from other Windows applications.

OODML

Use the Field object's *replaceFromFile*( ) method.

REPLACE OLE

Topic group Related topics

Inserts an OLE document into an OLE field.

Syntax

REPLACE OLE <OLE field name> FROM <filename>
[LINK]

<OLE field name>

The field where an OLE document is inserted.

FROM <file name>

The file that identifies an OLE document, including its extension. There is no default extension.

LINK

LINK stores a pointer to the OLE document. By default, *dBASE Plus* embeds the OLE document itself in the specified memo field.

Description

Use REPLACE OLE to insert the contents of an OLE document into an OLE field. You can either embed the actual OLE document in an OLE field (the default) or access the OLE document by linking it to the OLE field.

If you link the OLE document, the OLE field contains only a reference to the OLE document. As long as the OLE document remains in the same location, the OLE field displays the most current version of the document.

If you embed the OLE document, the OLE field contains a copy of the document. There are no links between the field and the OLE document: therefore, any changes to the original version of the OLE document are not reflected in the embedded document.

OODML

Use the Field object's *replaceFromFile*( ) method. The file is embedded; you cannot link it.

RLOCK( )

Topic group Related topics

Locks the current record or a specified list of records in the current or specified table.

Syntax

RLOCK([<list expC>, <alias>] | [<alias>])

<list expC>

The list of record numbers to lock, separated by commas. If omitted, the current record is locked.

<alias>

The work area to lock.

You don't have to specify record numbers if you want to specify a value for <alias> only. However, if you have specified record numbers, you must specify an <alias>.

Description

Use RLOCK( ) to lock the current record or a list of records in any open table. If you don't pass RLOCK( ) any arguments, it locks the current record in the current table. If you pass only <alias> to RLOCK( ), it locks the current record in the specified table. If RLOCK( ) is successful in locking all the records you specify, it returns *true*; otherwise it returns *false*. You can lock up to 100 records in each table open at your workstation with RLOCK( ).

You can view and update a record you lock with RLOCK( ). Other users can view this record but can't update it. When you lock a record with RLOCK( ), it remains locked until you do one of the following:

Issue UNLOCK
Close the table

All commands that change table data cause *dBASE Plus* to attempt to execute an automatic record or file lock. If *dBASE Plus* fails to execute an automatic record or file lock, it an error occurs. You might want to use RLOCK( ) to handle a lock failure youself, instead of letting the error occur.

RLOCK( ) can't lock the records you specify when any of the following conditions exist:

Another user has locked, explicitly or automatically, the current record or one of the records in <list expC>.
Another user has locked, explicitly or automatically, the table that contains the records you're trying to lock.

When SET REPROCESS is set to 0 (the default) and RLOCK( ) can't immediately lock its records, *dBASE Plus* prompts you to attempt the lock again or cancel the attempt. Until you choose to cancel the function, RLOCK( ) repeatedly attempts to get the record locks. Use SET REPROCESS to eliminate being prompted to cancel the RLOCK( ) function, or to set the number of locking attempts.

RLOCK( ) is similar to FLOCK( ), except FLOCK( ) locks an entire table. Use FLOCK( ), therefore, when you need to have sole access to an entire table or related tables—for example,

when you need to update multiple tables related by a common key—or when you want to update more than 100 records at a time.

When you set a relation in a parent table with SET RELATION and then lock a record in that table with RLOCK( ), *dBASE Plus* attempts to lock all child records in child tables. For more information on relating tables, see SET RELATION.

RLOCK( ) is equivalent to LOCK( ).

OODML

Use the Rowset object's *lockRow*( ) method.

ROLLBACK( )

Topic group Related topics

Cancels the transaction by undoing all logged changes.

Syntax

ROLLBACK([<database name expC>])

<database name expC>

The name of the database in which to rollback the transaction.

If you began the transaction with BEGINTRANS(<database name expC>), you must issue ROLLBACK(<database name expC>). If instead you issue ROLLBACK( ), *dBASE Plus* ignores the ROLLBACK( ) statement.

If you began the transaction with BEGINTRANS( ), <database name expC> is an optional ROLLBACK( ) argument. If you include it, it must refer to the same database as the SET DATABASE TO statement that preceded BEGINTRANS( ).

Description

A transaction works by logging all changes. If an error occurs while attempting one of the changes, or the changes need to be undone for some other reason, the transaction is canceled by calling ROLLBACK( ). Otherwise, COMMIT( ) is called to clear the transaction log, thereby indicating that all the changes in the transaction were committed and that the transaction as a whole was posted.

Since new rows have already been written to disk, rows that were added during the transaction are deleted. In the case of DBF tables, the rows are marked as deleted, but are not physically removed from the table. If you want to actually remove them, you can pack the table with PACK. Rows that were just edited are returned to their saved values.

All locks made during a transaction are maintained until the transaction is completed. This ensures that no one else can make any changes until the transaction is committed or abandoned.

For more information on transactions, see BEGINTRANS( ).

OODML

Call the *rollback*( ) method of the Database object.

SCAN

Topic group Related topics

Steps through each record in the current table, executing specified statements on each record that meets specified conditions.

Syntax

```
SCAN [<scope>] [FOR <condition 1>] [WHILE <condition 2>]
  [<statements>]
ENDSCAN
```

<scope>

FOR <condition 1>
WHILE <condition 2>

The scope of the command. The default scope is ALL.

<statements>

Statements to execute for each record visited.

ENDSCAN

A required keyword that marks the end of the SCAN loop.

Description

Use SCAN to process the current table record by record. With no scope options, SCAN starts with the first record in the table in the current index order and visits all the records, stopping at the end-of-file. You may specify a different <scope> and a WHILE condition to control the range of records, and a FOR condition that each record must pass for the <statements> to be executed.

At the end of each loop, *dBASE Plus*. automatically moves the record pointer forward one record in the table before returning to the beginning of the loop; therefore, don't include a SKIP command. You may use the EXIT command to exit out of the loop and the LOOP command to go to the next record, skipping all remaining commands in the loop.

You may nest SCAN loops, except that you cannot nest SCAN loops for the same table.

SCAN works like a DO WHILE .NOT. EOF( )...SKIP...ENDDO construct; however, with SCAN you can specify conditions with FOR, WHILE, and <scope>. SCAN also requires fewer lines of code than DO WHILE.

When using SCAN with an indexed table, don't change the value of a field that is (or is part of) the master index key. When you change the value of such a field, *dBASE Plus* repositions the record in the index file, which might cause unintended results. For example, if you change a key field that causes its record to move to the end of the index, that record might have the SCAN...ENDSCAN statements executed on it a second time.

If you change work areas within a SCAN loop, select the work area containing the table being scanned before control passes back to the first statement in the SCAN loop.

OODML

This example opens a table named FOO and traverses all the records, copying the value of the character field C1 to a throw-away variable, using a SCAN loop and the OODML equivalent:

```
use FOO
scan
x = C1
endscan
use
local q, r
q = new Query( "select * from FOO" )
r = q.rowset
if r.first( )
do
x = r.fields[ "C1" ].value
until not r.next( )
```

```
endif
```

Note:

A "SELECT * FROM" query is equivalent to a plain USE command.

A reference to the query's rowset is assigned to another variable as shorthand. It also executes a bit quicker.

A SCAN—without any scope qualifiers like REST or NEXT—always starts at the beginning of the table, so a call to the *first*( ) method is needed.

If *first*( ) returns false, there's nothing to do

A DO...UNTIL loop is used so that the navigation happens after processing the current row. Since the *first*( ) method returned true to get into the loop, there must be at least one row to process.

When *next*( ) returns false, you've hit the EOF, which terminates the loop.

SEEK

Topic group Related topics

Searches for the first record in an indexed table whose key fields matches the specified expression or expression list.

Syntax

SEEK <exp> | <exp list>

<exp>

The expression to search for in an index for a DBF table.

<exp list>

One or more expressions, separated by commas, to search for in a simple or composite key index for non-DBF tables.

Description

dBASE Plus can search a table for specific information either by a sequential search of a table or by an indexed search of the table's master index. A sequential search is similar to looking for information in a book by reading the first page, then the second, and so on, until the information is found or all pages have been read. LOCATE uses this method, checking each record until the information is found or the last record has been inspected.

An indexed search is similar to looking up a topic in a book index and turning directly to the appropriate page. Once a table index is created, SEEK can use this index to quickly identify the appropriate record.

SEEK looks for the first match in the index. If a match is found, the record pointer of the associated table is positioned at the record containing the match, and FOUND( ) returns *true*.

Use SKIP to access other records whose key fields match the index key fields or expression. SKIP advances the record pointer one record; because of the indexed order, other matches immediately follow the first. However, SKIP after SEEK (unlike CONTINUE after LOCATE) doesn't search for a match; it moves the record pointer one record whether or not it finds a match. You can combine SEEK and LOCATE or SEEK and SCAN (both with the WHILE clause) to do a quick indexed search for the first matching record before looking through or processing all the other matches.

The SET NEAR setting determines whether *dBASE Plus*, after an unsuccessful SEEK, positions the record pointer at the end-of-file or at the record in the indexed table immediately after the position at which the value searched for would have been found. If SET NEAR is OFF (the default) and SEEK is unsuccessful, EOF( ) returns *true* and FOUND( ) returns *false*. If SET NEAR is ON and SEEK is unsuccessful, EOF( ) returns *false* (unless the position at which the

sought value would have been found is the last record in the index), and FOUND( ) returns *false*.

The expression you look for with SEEK must match the key expression or fields of the master index. For example, if the master index key uses UPPER( ), the search expression must also be in uppercase.

For tables that support composite key indexes based on multiple fields, specify a value for each field in the composite key, separated by commas.

When you seek a key expression of type character, the rules established by SET EXACT determine if a match exists. If SET EXACT is OFF (the default) only the beginning characters of the key field need to be used for SEEK to find a match. For example, if SET EXACT is OFF, SEEK "S" will find "Sanders", or whatever the first key value is that starts with "S". If SET EXACT is ON, the expression must be identical to the key field for a match to exist.

SEEK and LOCATE each have their own advantages. SEEK conducts the most rapid searches; however, it requires an indexed table and can search only for values of the key expression.

If the information for which you are searching is in an unindexed table or is not contained in the key fields of an index, you can use LOCATE. LOCATE accepts any logical condition, which can specify any fields in the table in any combination. For large tables, however, a sequential search using LOCATE can be slow. In such cases, you might want to use INDEX to create a new index and then use SEEK or SEEK( ).

The SEEK( ) function works like SEEK followed by FOUND( ), except that SEEK searches in the current work area, while SEEK( ) can search in the current or a specified work area. However, SEEK( ) can only search for a single expression; it does not support composite keys based on multiple fields. SEEK( ) returns *true* or *false* depending on whether the search is successful.

OODML

Use the Rowset object's *findKey*( ) or *findKeyNearest*( ) methods.

SEEK( )

Topic group Related topics Example

Searches for the first record in an indexed table whose key matches the specified expression.

Syntax

SEEK(<exp> [,<alias>])

<exp>

The key value to search for.

<alias>

The work area you want to search

Description

SEEK( ) evaluates the expression <exp> and attempts to find its value in the master index of the table opened in the current or specified work area. SEEK( ) returns *true* if it finds a match of the key expression in the master index, and *false* if no match is found.

The SEEK( ) function combines the SEEK command and the FOUND( ) function, adding the ability to search in any work area. However, SEEK( ) does not support composite key indexes based on multiple fields used by non-DBF tables.

Because an index search is almost always followed by a test to see if the search was successful, when searching DBF tables, use SEEK( ) instead of SEEK and FOUND( ). FOUND( ) will return the result of the last SEEK( ) as well.

SET NEAR and SET EXACT affect SEEK( ) the same way they affect SEEK. See SEEK for more details.

OODML

Use the Rowset object's *findKey*( ) or *findKeyNearest*( ) methods.

SELECT

Topic group Related topics

Sets the current work area in which to open or perform operations on a table.

Syntax

SELECT <alias>

<alias>

The work area to select.

Description

Use SELECT to choose a work area in which to open a table, or to specify a work area in which a table is already open. Many commands operate on the currently selected work area only, or by default.

To select a table that is already open, its alias name is preferred over the work area number, because tables may be opened in different work areas depending on conditions. The alias name will always select the right table (or cause an error if the table is not opened), while the work area number may take you to the wrong table.

Each work area supports its own value of FOUND( ) and an independent record pointer. Changes in the record pointer of the active work area have no effect on the record pointers of any other work areas, unless you set a relation between the work areas with the SET RELATION command.

If the <alias> is in a variable, use the parentheses as indirection operators. For example, if xAlias is a variable containing a work area number or alias name, use SELECT(xAlias). Otherwise, *dBASE Plus* will attempt select a work area with the alias name "xAlias".

OODML

There is no concept of the "current" Query object. Use your usual object management techniques to manage Query objects.

SELECT( )

Topic group Related topics Example

Returns the number of an available work area or the work area number associated with a specified alias.

Syntax

SELECT([<alias>])

<alias>

The work area to return. (If <alias> is a work area number, there is no need to call this function, because that number is what the function will return.)

Description

If you do not specify an alias, SELECT( ) returns the number of the next available work area, a number between 1 and 225; or zero if all work areas in the current workset are being used. If you specify an alias, SELECT( ) determines whether the specified alias name is already in use. If it is, SELECT( ) returns the work area number that's using the alias name; otherwise it returns zero.

Use SELECT( ) to locate an available work area in which to open a table, or to see if a table is already open so that you don't open it again.

OODML

There is no concept of the "current" Query object. Use your usual object management techniques to manage Query objects.

SET AUTOSAVE

Topic group Related topics

Determines if *dBASE Plus* writes data to disk each time a record is changed or added.

Syntax

SET AUTOSAVE on | OFF

Description

Use SET AUTOSAVE ON to reduce the chances of data loss. When SET AUTOSAVE is ON and you alter or add a record, *dBASE Plus* updates tables and index files on disk when you move the record pointer. When SET AUTOSAVE is OFF, changes are saved to disk as the record buffer is filled.

Since *dBASE Plus* periodically saves table changes to disk, in most situations you don't need to SET AUTOSAVE ON. SET AUTOSAVE OFF lets you process data faster, since *dBASE Plus* writes your changes to disk less often.

OODML

AUTOSAVE is always OFF. To force data to be written to disk, call the Rowset object's *flush*( ) method in the *onSave* event.

SET DATABASE

Topic group Related topics

Sets the default database from which tables are accessed.

Syntax

SET DATABASE TO [<database name>]

<database name>

Specifies the name of the database you want to make the current database.

Description

SET DATABASE sets the current database, which defines the default location for tables accessed by dBL commands. Using this command, you can select from any databases previously opened with the OPEN DATABASE command. Databases are defined using the BDE Administrator.

When you issue the SET DATABASE TO command without a database, *dBASE Plus* restores operation to accessing tables in the current directory (or in the directory specified by SET PATH) instead of through a database.

OODML

Assign the appropriate Database object to the Query object's *database* property.

SET DBTYPE

Topic group Related topics

Sets the default table type to either Paradox or dBASE.

Syntax

SET DBTYPE TO [PARADOX | DBASE]

PARADOX | DBASE

Sets the default table type to a Paradox (DB) or dBASE (DBF) table. The default is DBASE.

Description

SET DBTYPE sets the default type of table used by commands that open or create a table. You can override this selection by specifying a specific extension, that is, .DBF for a dBASE table or .DB for a Paradox table; or by using the TYPE option offered by many commands.

SET DBTYPE TO specified without a DBASE or PARADOX argument returns DBTYPE to its default (DBASE).

OODML

No equivalent.

SET DELETED

Topic group Related topics

Controls whether *dBASE Plus* hides records marked as deleted in a DBF table.

Syntax

SET DELETED ON | off

Description

Use SET DELETED to include or exclude records marked as deleted in a DBF table. When SET DELETED is OFF, all records appear in a table. When SET DELETED is ON, *dBASE Plus* excludes records that have been marked as deleted, hiding them from most operations.

INDEX, REINDEX, and RECCOUNT( ) aren't affected by SET DELETED.

If two tables are related with SET RELATION, SET DELETED ON suppresses the display of deleted records in the child table. The related record in the parent table still appears, however, unless the parent record is also deleted.

OODML

Soft deletes are not supported.

SET EXACT

Topic group Related topics

Establishes the rules used to determine whether two character strings are equal.

Syntax

SET EXACT on | OFF

Description

Use SET EXACT to choose between a partial string match and an exact string match for certain Xbase DML commands, the Array class scan( ) method, and the = comparison operator. The == comparison operator always behaves like SET EXACT is ON.

When SET EXACT is OFF, partial string matches are performed. For example, SEEK("S") will find "Smith" in an index, and "Smith"="S" evaluates to *true*.

When SET EXACT is ON, the two strings must match exactly, except that trailing blanks are ignored. For example, SEEK("Smith") will find "Smith " in an index and "Smith"="Smith " will evaluate to *true*.

A partial string match can be thought of as a "begins with" check. For example, the SEEK( ) function searches for an index key value that begins with certain characters, and the = operator checks to see if one string begins with another string.

In language drivers that have primary and secondary weights for characters (not U.S. language drivers but most others), *dBASE Plus* compares characters by their primary weights when SET EXACT is OFF and by their secondary weights when SET EXACT is ON. For example, when SET EXACT is OFF, and the current language driver is German, "drücken" and "drucken" are equal.

OODML

Use the rowset object's *exactMatch* property.

SET EXCLUSIVE

Topic group Related topics

Controls whether *dBASE Plus* opens tables and their associated index and memo files in exclusive or shared mode.

Syntax

SET EXCLUSIVE on | OFF

Description

When you issue SET EXCLUSIVE ON, subsequent tables you open—and their associated indexes and memos—are in exclusive mode, unless you open them with USE...SHARED. When you open a table in exclusive mode, other users can't open, view, or change the file or any of its associated index and memo files. If you try to open a table that another user has opened in exclusive mode, or if you try to open in exclusive mode a table that another user has opened, an error occurs.

Exclusive use of a table is different than a file lock that you would get with FLOCK( ). With a file lock, others may have the table open and can read data, although only one user may have a file lock at any time. With exclusive use, no one else can have the table open.

SET EXCLUSIVE OFF causes subsequent tables you open—and their associated indexes and memos—to be in shared mode, unless you open them with USE...EXCLUSIVE. If a table in shared mode is in a shared network directory, other users on the network with access to the directory can open, view, and change the file and any of its associated index and memo files.

If you use SET INDEX and the table is open in exclusive mode, *dBASE Plus* opens the index in exclusive mode. If the table is open in shared mode by way of an overriding USE...SHARED, *dBASE Plus* opens the index in the mode specified by USE.

An index created with INDEX is opened in exclusive mode, regardless of whether the table is opened in shared or exclusive mode and regardless of the SET EXCLUSIVE setting. After creating an index, you can open the index in shared mode with USE...INDEX...SHARED or by issuing SET EXCLUSIVE OFF followed by SET INDEX TO.

The following commands require the exclusive use of a table with either SET EXCLUSIVE ON or USE...EXCLUSIVE:

CONVERT
DELETE TAG
INDEX...TAG
MODIFY STRUCTURE
PACK
REINDEX
ZAP

OODML

EXCLUSIVE is always OFF. When a method like *packTable*( ) requires exclusive access to a table, the method always attempts to open the table in exclusive mode. If the table is already open in another query, the method will fail.

SET FIELDS

Topic group Related topics Example

Defines the list of fields a table appears to have.

Syntax

SET FIELDS TO
[<field list> | ALL [LIKE <skeleton 1>] [EXCEPT <skeleton 2>]]

SET FIELDS on | OFF

<field list> | ALL [LIKE <skeleton 1> | EXCEPT <skeleton 2>]

Adds the specified fields to the list of fields the table appears to have. The fields list may include fields from tables open in all work areas and may also include read-only calculated fields. The following table provides a description of SET FIELDS TO options:

Option	Description
ALL	Adds all fields in the current work area to the field list
LIKE <skeleton 1>	Adds all fields in the current work area whose names match <skeleton 1> to the field list
EXCEPT <skeleton 2>	Adds all fields in the current work area except those whose

	names match <skeleton 2> to the field list
LIKE <skeleton 1> EXCEPT <skeleton 2>	Adds all fields in the current work area whose names are like <skeleton 1> except those whose names match <skeleton 2> to the field list

Description

When there is no field list, or SET FIELDS is OFF, operations in a work area that use all fields (by default) use all the fields in the currently selected table. For example, if you LIST a Customer table with 10 fields and 500 records, those 10 fields are displayed.

A field list overrides this default behavior, making the table appear to have the fields you specify. This is usually done to either:

- Restrict the fields to certain fields in the table. For example, you can make the Customer table appear to have only a customer ID and name, hiding the other 8 fields.
- Include fields in other related tables. For example, you could set a relation to a table of sales people, and make the Customer table appear to have the customer ID, customer name, and the name of their account representative.

When a field list is active, it is the field list for all work areas in the workset. Because the fields in a field list always contain the full alias and field name, the fields in the field list will always be used, even if those fields are in another, unrelated work area. For example, suppose you create a field list with fields from the Customer table, and then select the Vendor, which has 90 records and is not related into the Customer table. If you then issue LIST, you will see 90 records, because the LIST command works on the current work area, but you will see the fields from the Customer table—the fields from the same record repeated 90 times, because the two tables are not related, and those are the values of the named fields for each record in the Vendor table.

Therefore, when you create a field list, it is usually used only for the work area in which it is created. If the field list contains fields from other work areas, some way of synchronizing the movement of the record pointers in those work areas, usually with SET RELATION, is required.

If there is no field list, SET FIELDS creates the specified field list and activates it. If there is a field list, whether it's active or not, SET FIELDS adds the specified fields to the field list, and activates it. Fields in other work areas that are added to the field list may be referred to by their field name only, without using an alias; those fields now appear to be fields in the current work area. The alias is still allowed, and is necessary if you have two fields with the same name from different tables in the field list.

A field may be added to the field list more than once, although this is not recommended. For example, if you execute SET FIELDS TO ALL twice, you will see all the fields in the current table twice. Be sure to use CLEAR FIELDS before issuing SET FIELDS if your intent is to create a new field list, not add to an existing field list. To specify a field in other work areas, prefix the field name with the alias name and alias operator (that is, alias->field).

You can temporarily disable the field list with SET FIELDS OFF. To reactivate the field list, use SET FIELDS ON. Adding fields with SET FIELDS always reactivates the field list. If you SET FIELDS ON without using the SET FIELDS TO <field list> command, no fields are accessible. SET FIELDS TO with no fields has the same effect as CLEAR FIELDS, deactivating and clearing the field list.

Some commands have a FIELDS option, or some way of specifying fields. You may further restrict the fields used with this option, but you cannot reference fields that have been hidden because they have not been included with SET FIELDS.

When a field list is active, fields that are not on the field list cannot be used in expressions. However, some commands ignore the field list, including:

- INDEX and index expressions
- LOCATE

SET FILTER
SET RELATION

The fields list specified with SET FIELDS TO can include both table field names and calculated fields. The /R option provides a setting to specify read-only access to table fields, for example:

```
salary/R, hours/R
```

To specify a calculated field, you can specify any valid expression. For example,

```
gross_pay = salary * hours
```

OODML

No direct equivalent. When accessing the *fields* array, you may include program logic to include or exclude specific fields.

SET FILTER

Topic group Related topics Example

Hides records based on a logical condition.

Syntax

SET FILTER TO [<condition>]

<condition>

The condition that records must meet to be seen.

Description

A filter is a mechanism by which you can temporarily hide, or filter out, those records that do not match certain criteria so that you can see only those records that do match. The criteria is expressed as a logical expression, for example,

```
set filter to upper( FIRST_NAME ) == "WALDO"
```

In this case, you would see only those records in the current table whose First_name field was "Waldo" (capitalized in any way).

The filter does not take effect until some sort of record navigation is attempted. For example, any command with an ALL scope will attempt to start at the first record. In this case, the command will start at the first record that matches the filter condition, and process all matching records. A SKIP command would attempt to navigate to the next matching record, and SKIP -1 would attempt to navigate to the previous matching record. GO TOP is often used after SET FILTER to position the record pointer on the first matching record.

Any time you attempt to navigate to a record, the record is evaluated to see if it matches the filter condition. If it does, then the record pointer is allowed to position itself at that record and the record can be seen. If the record does not match the filter condition, the record pointer continues in the direction it was moving to find the next matching record. It will continue to move in that direction until it finds a match or reaches end-of-file. For example, suppose you issue:

```
skip 4
```

If no filter is active, you would move four records forward. If a filter is active, the records pointer will move forward until it has encountered four records that match the filter condition, and stop at the fourth. That may be the next four records in the table, if they all happen to match, or the next five, or the next 400, or never, if there aren’t four records after the current record that match. In that last case, the record pointer will be at the end-of-file.

In other words, when there is no filter active, every record is considered a match. By setting a filter, you filter out all the records that don't match certain criteria.

Note

You cannot use the special variables *this* or *form* in the <condition>. This is explicitly prohibited because these special variables automatically take on the value of whatever object and form has focus (or fires an event) at any given moment. Therefore, the filter condition will vary and quite likely be invalid. Generally speaking, you should not use variables in a filter condition at all, because the variables may go out of scope, making the filter condition an invalid expression. To solve these problems, use macro substitution, as shown in the example.

Many commands have scope option that includes FOR and WHILE conditions. These conditions are applied in addition to the filter condition.

SET FILTER applies to the current work area. Each work area may have its own filter condition. To disable the filter condition, issue SET FILTER TO with no options.

OODML

Use the rowset object's *beginFilter*( ) and *applyFilter*( ) methods.

SET HEADINGS

Topic group Related topics

Controls the display of field names in the output of DISPLAY and LIST.

Syntax

SET HEADINGS ON | off

Description

When SET HEADINGS is ON, the output of DISPLAY and LIST includes a heading identifying the fields of the table(s). Issue SET HEADINGS OFF before issuing DISPLAY or LIST to view the output without field-name headings.

OODML

No equivalent.

SET INDEX

Topic group Related topics

Opens index files for the current DBF table.

Syntax

SET INDEX TO [<filename list> [ORDER [TAG]
 <ndx filename> | <tag name> [OF <mdx filename>]]]

<filename list>

Specifies the index files to open, including both .NDX and .MDX indexes. The default extension is .MDX.

ORDER [TAG] <ndx filename> | <tag name>

Specifies a master index, which can be an .NDX file or a tag name contained within an .MDX index file. The TAG keyword is for readability only; it has no effect on the command.

OF <mdx filename>

Specifies a multiple index file containing <tag name>. The default extension is .MDX.

Description

Use SET INDEX to open the specified .NDX and .MDX files in the current work area. Open index files are updated when changes are made to the associated table. Including an index file list when issuing USE...INDEX is equivalent to following the USE command with the SET INDEX command.

If the first index opened with SET INDEX is an .NDX file, that index becomes the master index unless you specify another master index with the ORDER option or the SET ORDER command. If the first index opened with SET INDEX is an .MDX file and you don't specify the ORDER clause, no master index is defined, and records in the table appear in record number or natural order. To specify a master index for the current table, specify the ORDER option or use the SET ORDER command.

Before opening the indexes specified with the command, SET INDEX closes all open index files except the production .MDX file, the index file with the same name as the current table. Specifying SET INDEX TO without a list of indexes closes all open .NDX and .MDX files in the current work area, except for the production index file. You can also use the CLOSE INDEX command. All indexes, including the production .MDX file, are closed when you close the table.

The order in which you specify indexes with the SET INDEX command isn't necessarily the same as the order *dBASE Plus* uses for functions like TAG( ). Open indexes for a specified work area are ordered as follows:

1. All .NDX index files in the order you list them in <filename list>.
    - All tags in the production .MDX file, in the order they were created. The tags are listed in order by the DISPLAY STATUS command.
    - All tags in other open .MDX files.

The order of the open indexes remains the same until you specify another index order with the USE...INDEX or SET INDEX commands, or you issue an INDEX command.

OODML

Assign to the Rowset object's *indexName* property.

SET KEY TO

Topic group Related topics

Constrains the records in a table to those whose key field values falls within a range.

Syntax

```
SET KEY TO
 [ <exp1> | <exp list 1> |
 RANGE
  <exp2> [,] | ,<exp3> | <exp2>, <exp 3>
  [EXCLUDE] |
 LOW <exp list 2>] [HIGH <exp list 3>]
  [EXCLUDE] ]
[IN <alias>]
```

<exp1>

Shows only those records whose key value matches <exp 1>.

<exp list 1>

For tables indexed on a composite (multi-field) key index, shows only those records whose key field values match the corresponding values in <exp list 1>, separated by commas.

RANGE <exp2> [,] | ,<exp3> | <exp2>, <exp3>
LOW <exp list 2> HIGH <exp list 3>

Shows only those records whose key values fall within a range. Use RANGE for single key values and LOW/HIGH for composite keys. You may use either the RANGE clause or LOW/HIGH, but not both in the same command. The following table summarizes how SET KEY constrains records in the master index:

Option	Description
RANGE <exp2> [,] LOW <exp list 2>	Shows only those records whose key values are greater than or equal to <exp2>/<exp list 2>
RANGE , <exp3> HIGH <exp list 3>	Shows only those records whose key values are less than or equal to <exp3>/<exp list 3>
RANGE <exp2>, <exp3> LOW <exp list 2> HIGH <exp list 3>	Shows only those records whose key values are greater than or equal to <exp2>/<exp list 2> and less than or equal to <exp3>/<exp list 3>

EXCLUDE

Excludes records whose key values are equal to <exp2>/<exp list 2> or <exp3>/<exp list 3>. If omitted, these records are included in the range.

IN <alias>

The work area in which to set the key constraint.

Description

SET KEY TO is similar to SET FILTER; SET KEY TO uses the table's current master index and shows only those records whose key value matches a single value or falls within a range of values. This is referred to as a key constraint. Because it uses an index, a key constraint is instantaneous, while a filter condition must be evaluated for each record. SET KEY TO with no arguments removes any range of key values previously established for the current table with SET KEY TO.

The key range values must match the key expression of the master index. For example, if the index key is UPPER(Name), specify uppercase letters in the range expressions. In determining whether the specified range expressions match key field expressions, SET KEY TO follows the rules established by SET EXACT. The SET KEY TO range takes effect after you move the record pointer.

When you issue both SET KEY and SET FILTER for the same table, *dBASE Plus* processes only records that are within the SET KEY index range and that also meet the SET FILTER condition.

OODML

Use the Rowset object's *setRange*( ) method.

SET LOCK

Topic group Related topics

Determines whether *dBASE Plus* attempts to lock a shared table during execution of certain commands that read the table but don't change its data.

Syntax

SET LOCK ON | off

Description

Issue SET LOCK OFF to disable automatic file locking for certain commands that only read a table. This lets other users change data in the file while you access it with read-only commands. For example, you might want to issue SET LOCK OFF before using AVERAGE if you don't expect other users to alter the data in the table you're using significantly. Or, you might want to issue SET LOCK OFF before processing a range of records that other users aren't going to update.

The following commands automatically lock tables when SET LOCK is ON and don't lock tables when SET LOCK is OFF:

- AVERAGE
- CALCULATE
- COPY (source file)
- COPY MEMO
- COPY STRUCTURE
- COPY TO ARRAY
- COPY STRUCTURE [EXTENDED] (source file)
- COUNT
- SORT (source file)
- SUM
- TOTAL (source file)

dBASE Plus continues to lock records and tables automatically for commands that let you change data whether SET LOCK is ON or OFF.

OODML

This setting is not applicable.

SET MEMOWIDTH

Topic group Related topics

Sets the width of memo field display or output.

Syntax

SET MEMOWIDTH TO [<expN>]

<expN>

Specifies a number from 8 to 255 that sets the width of memo field display and output. The default is 50.

Description

Use SET MEMOWIDTH to change the column width of memo fields during display and output, and the default column width for the MEMLINES( ) and MLINE( ) functions. Memo fields can be displayed using the commands DISPLAY, LIST, ?, or ??. SET MEMOWIDTH doesn't affect the display of a memo field in an Editor control. If the system memory variable variable_wrap is set to *true*, the system memory variables _lmargin and _rmargin determine the memo width.

The @V (vertical stretch) picture function causes memo fields to be displayed in a vertical column when _wrap is *true*. When @V is specified, the _pcolno system memory variable is incremented by the @V value. This lets you change the appearance of the printed output of ? or

?? commands by using the @V function. When @V is equal to zero, memo fields wrap within the SET MEMOWIDTH width.

OODML

This setting is not applicable.

SET NEAR

Topic group Related topics

Specifies where to move the record pointer after a SEEK or SEEK( ) operation fails to find an exact match.

Syntax

SET NEAR on | OFF

Description

Use SET NEAR to position the record pointer in an indexed table close to a particular key value when a search does not find an exact match. When SET NEAR is ON, the record pointer is set to the record closest to the key expression searched for but not found with SEEK or SEEK( ). When SET NEAR is OFF and a search is unsuccessful, the record pointer is positioned at the end-of-file.

The closest record is the record whose key value follows the value searched for in the index order, toward the end-of-file. When SET DELETED is ON or a filter is set with the SET FILTER command, SET NEAR skips over deleted or filtered-out records in determining the record nearest the key value expression. The record pointer will be at the end-of-file if search value comes after the key value of the last record in the index order.

With SET NEAR ON, FOUND( ) and SEEK( ) return *true* for an exact match or *false* for a near match. With SET NEAR OFF, FOUND( ) and SEEK( ) return *false* if no match occurs.

OODML

Use either the *findKey*( ) or *findKeyNearest*( ) method of the Rowset object.

SET ODOMETER

Topic group

Specifies how frequently *dBASE Plus* updates and displays record counter information for certain commands in the status bar.

Syntax

SET ODOMETER TO [<expN>]

<expN>

The interval at which *dBASE Plus* updates the record counter. <expN> must be at least 1 and is truncated to an integer. If omitted, the default value, 100, is used.

Description

Use SET ODOMETER to specify how frequently *dBASE Plus* updates and displays record counter information during the execution of certain commands, such as AVERAGE, CALCULATE, COUNT, DELETE, GENERATE, INDEX, RECALL, SUM, and TOTAL. If the

status bar is not enabled, SET ODOMETER has no effect. The status bar is enabled through _app.statusBar.

If SET TALK is OFF, *dBASE Plus* does not display any record counter information in the status bar, regardless of the SET ODOMETER setting. If SET TALK is ON, use SET ODOMETER with a relatively high value to improve performance.

OODML

Use the Session object's *onProgress* event.

SET ORDER

Topic group Related topics

Specifies an open index file or tag as the master index of a table.

Syntax

SET ORDER TO [[TAG] <tag name> [OF <mdx>] [NOSAVE]]

[TAG] <tag name>

The name of an index tag in an open .MDX file or the name of an open .NDX file (without the file extension). The TAG keyword is included for readability only; TAG has no affect on the operation of the command.

OF <mdx>

The open .MDX file that contains <tag name>. Use this option when two open .MDX files have a tag with the same name. The default extension is .MDX.

If you use the <tag name> option but don't specify <mdx>, *dBASE Plus* searches for the named index in the list of open indexes.

NOSAVE

Used to delete a temporary index after the associated table is closed. If you decide after choosing this option that you want to keep the index, open the index again using SET ORDER without the NOSAVE option, before you close the table.

Description

Use SET ORDER to change the master index of a table without having to close and reopen indexes. You can choose the master index from the list of .NDX files or .MDX index tags opened with the SET INDEX or USE...INDEX commands.

If you specify SET ORDER without specifying an index, the table appears in primary key order, if the table has a primary key; or unindexed, in record number order.

OODML

Assign the tag name to the Rowset object's *indexName* property.

SET REFRESH

Topic group Related topics

Determines how often *dBASE Plus* refreshes the workstation screen with table information from the server.

Syntax

SET REFRESH TO <expN>

<expN>

A time interval expressed in seconds from 0 to 3,600 (1 hour), inclusive. The default is 0, meaning that *dBASE Plus* doesn't update the screen.

Description

Use SET REFRESH to set a refresh interval when working with shared tables on a network. Then, when you use BROWSE or EDIT to edit shared tables, your screen refreshes at the interval you set, showing you changes made by other users on the network to the same tables.

If another user has a lock on the file or records you're currently viewing, the file or records won't be refreshed until that user releases the lock.

OODML

Use a Timer object to periodically call the Rowset object's *refreshControls*( ) method.

SET RELATION

Topic group Related topics Example

Links two or more open tables with common fields or expressions.

Syntax

```
SET RELATION TO
[<key exp list 1> | <expN 1>
 INTO <child table alias 1> [CONSTRAIN]
 [, <key exp list 2> | <expN 2>
  INTO <child table alias 2> [CONSTRAIN] ]
[ADDITIVE] ]
```

<key exp list 1>

The key expression or field list that is common to both the current table and a child table and links both tables. The child table must be indexed on the key field and that index must be the master index in use for the child table.

<expN 1> INTO <child table alias>

For dBASE tables only, you can specify <expN> to link records in a child table. When <expN> is RECNO(), dBASE Plus links the current table to a child table by corresponding record numbers, in which case the child table doesn't have to be indexed.

INTO <child table alias>

<alias> specifies the child table linked to the current table.

<key exp list 2> | <expN 2> INTO <alias 2> ...]

Specifies additional relationships from the current table into other tables.

CONSTRAIN

Limits records processed in the child table to those matching the key expression in the parent table.

ADDITIVE

Adds the new relation to any existing ones. Without ADDITIVE, SET RELATION clears existing relations before establishing the new relation.

Description

Use SET RELATION to establish a link between open tables based on common fields or expressions.

Before setting a relation, open each table in a separate work area. When a relation is set, the table in the current work area is referred to as the parent table, and a table linked to the parent table by the specified key is called a child table. The child table must be indexed on the fields or expressions that link tables and that index must be the master index in use for the child table.

A relation between tables is usually set through common keys specified by <key exp list>. The relating expression can be any expression derived from the parent table that matches the keys of the child table master index. The keys may be a single field or a set of concatenated fields contained in each table. The fields in each table can have different names but must contain the same type of data. For Paradox and SQL tables, you can specify single or composite index key fields.

SET RELATION clears existing relations before establishing a new one, unless you use the ADDITIVE option. SET RELATION TO without any arguments clears existing relations from the current table without establishing any new relations.

The CONSTRAIN option restricts access in the child table to only those records whose key values match records in a parent table. This is the same as using SET KEY TO on the key field of the child table. As a result, you can't use SET KEY TO and CONSTRAIN at the same time. If a SET KEY TO operation is in effect on the child table when you specify CONSTRAIN with SET RELATION, *dBASE Plus* returns a "SET KEY active in alias" message. If the CONSTRAIN option is in effect when SET KEY TO is specified, *dBASE Plus* returns the error "Relation using CONSTRAIN." You can use SET FILTER with the CONSTRAIN option, if you want to specify additional conditions to qualify records in a child table.

More than one relation can be defined from the same table. Also, more than one relation can be set from the same parent table if you use the ADDITIVE option or if you specify multiple relations with the same SET RELATION command. You can also establish additional relations from a child table, thus defining a chain of relations. Cyclic relations aren't allowed; that is, *dBASE Plus* returns an error if you attempt to define a relation from a child table back into its parent table.

When a relation is set from a parent table to a child table, the relation can be accessed only from the work area that contains the parent table. To access fields of the child table from the current work area, use the alias operator(->) and prefix the name of fields in the child table by its alias name.

If a matching record can't be found in a linked table, the linked table is positioned at the end-of-file, and EOF( ) in the child alias returns *true* while FOUND( ) returns *false*. The setting of SET NEAR does not affect positioning of the record pointer in child tables.

When a SET SKIP list is active, the record pointer is advanced in each table, starting with the last work area in the relation chain and moving up the chain toward the parent table.

OODML

Use the Rowset object's *masterFields* and *masterRowset* properties, or the Query object's *masterSource* property.

SET REPROCESS

Topic group Related topics

Specifies the number of times *dBASE Plus* tries to lock a file or record before generating an error or returning *false*.

Syntax

SET REPROCESS TO <expN>

<expN>

A number from –1 to 32,000, inclusive, that is the number of times for *dBASE Plus* to try get a lock. The default is 0; both 0 and -1 have a special meaning, described below.

Description

Use SET REPROCESS to specify how many times *dBASE Plus* should try to get a lock before giving up. RLOCK( ) and FLOCK( ) return *false* if the lock attempt fails. For automatic locks, failure to get a lock causes an error. SET REPROCESS affects RLOCK( ), and FLOCK( ), and all commands and functions that automatically attempt to lock a file or records.

SET REPROCESS TO 0 causes *dBASE Plus* to display a dialog that gives you the option of canceling while it indefinitely attempts to get the lock.

Setting SET REPROCESS to a number greater than 0 causes *dBASE Plus* to retry getting a lock the specified number of times without prompting.

SET REPROCESS TO -1 causes *dBASE Plus* to retry getting a lock indefinitely, without prompting.

OODML

Set the Session object's *lockRetryCount* property.

SET SAFETY

Topic group Related topics

Determines whether *dBASE Plus* asks for confirmation before overwriting a file or removing records from a table when you issue ZAP.

Syntax

SET SAFETY ON | off

Description

When SET SAFETY is ON, *dBASE Plus*. prompts for confirmation before overwriting a file or removing records from a table when you issue ZAP. If you want your application to control the interaction between *dBASE Plus* and the user with regard to overwriting files, issue SET SAFETY OFF in your program.

SET SAFETY affects the following commands:

Commands using a TO FILE option
COPY
COPY FILE
COPY STRUCTURE [EXTENDED]
CREATE/MODIFY commands
INDEX
SAVE
SET ALTERNATE TO
SORT
TOTAL

UPDATE
ZAP

SET SAFETY also affects the PUTFILE( ) function

Note

SET TALK OFF does not suppress SET SAFETY warnings.

OODML

SAFETY is always OFF.

SET SKIP

Topic group Related topics

Specifies how to advance the record pointer through records of linked tables.

Syntax

SET SKIP TO [<alias1> [, <alias2>]...]

<alias1> [, <alias2>] ...

The work areas defined in the relation.

Description

SET SKIP works only with tables that have been linked with the SET RELATION command. Used together, the SET RELATION and SET SKIP commands determine the way in which the record pointer moves through parent and child tables.

Use SET SKIP when you set a relation from a parent table containing unique key values to child tables that contain duplicate key values, that is, a one-to-many relationship. SET SKIP causes commands that move the record pointer to move the pointer to every record with matching key values in a child table before moving the record pointer in the parent table. If you define a chain of relations and use SET SKIP to move from one table to the next down the chain, the record pointer moves to every record in the last child table before the pointer moves in its parent table.

You do not need to specify the root (parent) alias in the SET SKIP list. SET SKIP TO with no options cancels any previously defined SET SKIP behavior.

OODML

SET SKIP behavior in OODML is provided through use of the *navigateMaster* and *navigateByMaster* Rowset properties.

SET UNIQUE

Topic group Related topics

Determines if unique indexes are always created.

Syntax

SET UNIQUE on | OFF

Description

When SET UNIQUE is ON, the INDEX command always creates the index as if the UNIQUE option is specified. The UNIQUE option has different meanings for different table types. For

DBF tables, it allows records with duplicate key values to be stored in the table, but only shows the first record with that key value.

For DB and SQL tables, it prevents records with duplicate key values from being stored in the table; attempting to do so causes a key violation error. This type of index is referred to as a distinct index. You can create the same kind of index for DBF tables by using the DISTINCT option.

Whenever you reindex an index file, *dBASE Plus* maintains the index in the same way it was created. For more information on unique and distinct indexes, see the INDEX command.

OODML

No equivalent.

SET VIEW

Topic group Related topics

Opens a previously defined .QBE query or DBF table.

Syntax

SET VIEW TO <filename>

<filename>

The query or view file containing the commands to define the current working environment or view. If you specify a file without including its extension, *dBASE Plus* looks for a .QBE query or a .DBF table.

Description

Use SET VIEW to change the working environment of the current workset to one that was previously defined by CREATE QUERY or CREATE VIEW. The working environment includes open tables and index files, all relations, the active fields list, and filter conditions.

OODML

Use a DataModule object.

SKIP

Topic group Related topics

Moves the record pointer in the current or specified work area.

Syntax

SKIP [<expN>] [IN <alias>]

<expN>

The number of records *dBASE Plus* moves the record pointer forward or backward in the table open in the current or specified work area. If <expN> evaluates to a negative number, the record pointer moves backward. SKIP with no <expN> argument moves the record pointer forward one record.

IN <alias>

The work area in which to move the record pointer.

Description

Use SKIP to move the record pointer relative to its current position, in the current index order, if any.

If you issue a SKIP command when the record pointer is at the last record in a table, the record pointer moves to the end-of-file; EOF( ) returns *true*. Issuing any additional SKIP commands (that move forward) causes an error. Similarly, if you issue a SKIP -1 command when the record pointer is at the first record of a file, BOF( ) returns *true*, and a subsequent negative SKIP command cause an error.

SKIP IN <alias> lets you advance the record pointer in another work area without selecting that work area first with the SELECT command.

If you are using SKIP in a loop to visit all the records in a table, consider using a SCAN loop instead.

OODML

Call the *next*( ) method of the desired Rowset object.

SORT

Topic group Related topics Example

Copies the current table to a new table, arranging records in the specified order.

Syntax

```
SORT TO <filename> [[TYPE] PARADOX | DBASE]
ON <field 1> [/A | /D [/C]]
  [,<field 2> [/A | /D [/C]]...]
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]
[ASCENDING | DESCENDING]
```

<filename>

The new table file to copy and sort the current table's records to.

[TYPE] PARADOX | DBASE

Specifies the type of table you want to create, if <filename> does not include an extension. This option overrides the current SET DBTYPE setting. The TYPE keyword is included for readability only; it has no effect on the operation of the command.

ON <field 1>

Makes <field 1> the first field of <filename>and sorts <filename> records by the values in <field 1>, which can be any data type except binary, memo, or OLE.

/A

Sorts records in ascending order (A to Z; 1 to 9; past to future (blank dates come after non-blank dates); *false* then *true*). Since this is the default sort order, include /A for readability only.

/D

Sorts records in descending order.

/C

Removes the distinction between uppercase and lowercase letters. When you specify both A and C, or both D and C, use only one forward slash (for example, /DC).

<field 2> [/A | /D [/C]] ...

Sorts on a second field so that the new table is ordered first according to <field 1>, then, for identical values of <field 1>, according to <field 2>. If a third field is specified, records with identical values in <field 1> and in <field 2> are then sorted according to <field 3>. The sorting continues in this way for as many fields as are specified.

<scope>
FOR <condition 1>
WHILE <condition 2>

The scope of the command. The default scope is ALL.

ASCENDING

Sorts all specified fields for which you don't include a sort order in ascending order. Since this is the default, include ASCENDING for readability only.

DESCENDING

Sorts all specified fields for which you don't include a sort order in descending order.

Description

The SORT command creates a new table in which the records in the current table are positioned in the order of the specified key fields.

SORT creates a temporary index file. During the sorting process, your disk must have space for this temporary index file and the new table file.

SORT differs from INDEX in that it creates a new table rather than provide an index to the original table. Although using SORT is generally not as efficient as using an index to organize tables, you might want to use SORT for the following applications:

- To archive an outdated table and store it in a sorted order
- To create a table that is a sorted subset of an original table
- To maintain a small table that needs to be sorted in only one order
- To create an ordered table where record numbers are sequential and contiguous

OODML

No equivalent.

STORE AUTOMEM

Topic group Related topics

Stores the contents of all the current record's fields to a set of memory variables.

Syntax

STORE AUTOMEM

Description

STORE AUTOMEM copies every field of the current record to a set of matching automem variables. Each memory variable has the same name, length, and data type as one of the fields. *dBASE Plus* creates these memory variables if they don't already exist.

Automem variables let you temporarily store the data from table records, manipulate the data as memory variables rather than as field values, and then return the data to the table (using REPLACE AUTOMEM or APPEND AUTOMEM).

STORE AUTOMEM is one of three commands that create automem variables. The other two, USE <filename> AUTOMEM and CLEAR AUTOMEM, initialize blank automem variables for the fields of the current table.

When referring to the value of automem variables you need to prefix the name of an automem variable with M-> to distinguish the variable from the corresponding fields, which have the same name. The M-> prefix is not needed during variable assignment; the STORE command and the = and := operators do not work on Xbase fields.

OODML

The Rowset object's contains an array of Field objects, accessed through its *fields* property. These Field objects have *value* properties that may be programmed like variables.

SUM

Topic group Related topics

Computes a total for specified numeric fields in the current table.

Syntax

SUM [<*exp list*>]
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]
[TO <memvar list> | TO ARRAY <array>]

<exp list>

The numeric fields, or expressions involving numeric fields, to sum.

<scope>
FOR <condition 1>
WHILE <condition 2>

The scope of the command. The default scope is ALL.

TO <memvar list> | TO ARRAY <array>

Initializes and stores sums to the variables (or properties) of <memvar list> or stores sums to the existing array <array>. If you specify an array, each field sum is stored to elements in the order in which you specify the fields in <exp list>. If you don't specify <exp list>, each field sum is stored in field order. <array> can be a single- or multidimensional array; the array elements are accessed via their element numbers, not their subscripts.

Description

The SUM command computes the sum of numeric expressions and stores the results in specified variables or array elements. If you store the values in variables, the number of variables must be exactly the same as the number of fields or expressions summed. If you store the values in an array, the array must already exist, and the array must contain at least as many elements as the number of summed expressions.

If SET TALK is ON, SUM also displays its results in the result pane of the Command window. The SET DECIMALS setting determines the number of decimal places that SUM displays. Numeric fields in blank records are evaluated as zero. To exclude blank records, use the ISBLANK( ) function in defining a FOR condition. EMPTY( ) excludes records in which a specified expression is either 0 or blank.

SUM is similar to TOTAL, which operates on an indexed or sorted table to create a second table containing the sums of the numeric and float fields of records grouped on a key expression.

OODML

Loop through the rowset to calculate the sum.

TAG( )

Topic group Related topics Example

Returns the name of an open index.

Syntax

TAG([<.mdx filename expC>,] [<index number expN> [,<alias>]])

<.mdx filename expC>

The .MDX file that contains the index tag you want to check. The .MDX must be opened in the specified work area. If omitted, all open indexes, including the production .MDX file, are searched.

<index position expN>

the numeric position of the index tag in the specified .MDX file, or the position of the index in the list of open indexes.

<alias>

The work area you want to check.

Note

Unlike most functions, the first parameter is optional. If you omit the first parameter, the remaining parameters shift forward one; the second parameter becomes the first parameter, and so on.

Description

TAG( ) returns the name of the specified index, either:

- The tag name of an index in an .MDX file, or
- The name of an .NDX file, without the file extension.

The index must be referenced by number. If you do not specify an .MDX file to check, index numbering in the list of open indexes is complicated if you have open .NDX indexes or you have open non-production .MDX files. For more information on index numbering, see SET INDEX.

If you do not specify an index tag, TAG( ) returns the name of the current master index tag, or an empty string if there is no master index.

If the specified .MDX file or index tag does not exist, TAG( ) returns an empty string.

OODML

No equivalent.

TAGCOUNT( )

Topic group Related topics

Returns the number of active indexes in a specified work area or .MDX index file.

Syntax

TAGCOUNT([<.mdx filename> [,<alias>]])

<.mdx filename expC>

The .MDX file you want to check. The .MDX must be opened in the specified work area.

<alias>

The work area you want to check.

Description

TAGCOUNT( ) returns the total number of open indexes or the number of index tag names in a specified .MDX file. TAGCOUNT( ) returns 0 if there are no indexes or index tags open for the current or specified work area, or if the .MDX index file specified with <.mdx filename expC> does not exist. If you do not specify an .MDX file name, TAGCOUNT( ) returns the total number of indexes in the specified work area: the number of open .NDX files, plus the total number of tags in all open .MDX files. If you do not specify an alias, TAGCOUNT( ) returns the total number of indexes in the current work area.

OODML

No equivalent.

TAGNO( )

Topic group Related topics

Returns the index number of the specified index.

Syntax

TAGNO([<tag name expC> [,<.mdx filename expC> [,<alias>]]])

<tag name expC>

The name of the index tag that you want to return the position of. If you don't specify a tag name, TAGNO( ) returns the position of the current master index.

<.mdx filename expC>

The .MDX file that contains the index tag you want to check. The .MDX must be opened in the specified work area. If omitted, all open indexes, including the production .MDX file, are searched.

<alias>

The work area you want to check.

Description

TAGNO( ) returns a number that indicates the position of the specified index name in the list of open indexes in the current or specified work area. The order of indexes is determined by the order in which they were opened with the USE or SET INDEX commands.

If you don't specify a tag name, TAGNO( ) returns the number of the master index. If you don't specify an .MDX file name, TAGNO( ) searches the list of open index files in the specified work area, including .NDX files. If you don't specify an alias, TAGNO( ) operates on the list of open indexes in the current work area.

TAGNO( ) returns zero if the specified index tag or .MDX file does not exist.

Use TAGNO( ) to get the index number of an index when you know the tag name for functions like DESCENDING( ), FOR( ), KEY( ), and UNIQUE( ).

OODML

No equivalent.

TARGET( )

Topic group Related topics

Returns the name of a table linked with the SET RELATION command.

Syntax

TARGET(<expN> [,<alias>])

<expN>

The number of the relation that you want to check.

<alias>

The work area you want to check.

Description

TARGET( ) returns a string containing the name of the child tables that are linked to a parent table by the SET RELATION command. You must specify the number of the relation; if the table in the current or specified work area is linked to only one table, that <expN> is the number 1. TARGET( ) returns an empty string ("") if no relation is set in the <expN> position.

Use TARGET( ) to save the link tables of all SET RELATION settings for later use when restoring relations. To save the link expression, use the RELATION( ) function.

OODML

No equivalent. The *masterSource* and *masterRowset* properties contain references to the parent query or rowset; TARGET( ) returns the names of the child tables.

TOTAL

Topic group Related topics Example

Creates a table that stores totals for specified numeric fields of records grouped by common key values.

Syntax

TOTAL ON <key field> TO <filename> [[TYPE] PARADOX | DBASE]
[<scope>]
[FOR <condition 1>]
[WHILE <condition 2>]
[FIELDS <field list>]

<key field>

The name of the field on which the current table has been indexed or sorted.

TO <filename>

The table to create.

[TYPE] PARADOX | DBASE

Specifies the type of table you want to create, if <filename> does not include an extension. This option overrides the current SET DBTYPE setting. The TYPE keyword is included for readability only; it has no effect on the operation of the command.

<scope>
FOR <condition 1>
WHILE <condition 2>

The scope of the command. The default scope is ALL.

FIELDS <field list>

Specifies which numeric and float fields to total. If you don't include FIELDS, *dBASE Plus* totals all numeric and float fields.

Description

Use TOTAL to total the value of numeric fields in a table and create a second table to store the results. The numeric fields in the table storing the results contain totals for all records that have the same key field in the original table.

The current table must be either indexed or sorted on the key field. All records with the same key field become a single record in the table storing the result totals. All numeric fields appearing in the fields list contain totals. All other fields contain data from the first record of the set of records with identical keys.

To limit the fields that are created in the new file, or to group on more than one key field, use SET FIELDS as shown in the example.

TOTAL is similar to SUM, except that SUM operates on an indexed or unindexed table, returning a sum for all records of each numeric field. SUM doesn't create another table, but stores the results to memory variables or an array.

OODML

No equivalent.

UNIQUE( )

Topic group Related topics Example

Indicates if a specified index ignores duplicate records.

Syntax

UNIQUE([<.mdx filename expC>,] [<index position expN> [,<alias>]])

<.mdx filename expC>

The .MDX file that contains the index tag you want to check. The .MDX must be opened in the specified work area. If omitted, all open indexes, including the production .MDX file, are searched.

<index position expN>

the numeric position of the index tag in the specified .MDX file, or the position of the index in the list of open indexes.

<alias>

The work area you want to check.

Note

Unlike most functions, the first parameter is optional. If you omit the first parameter, the remaining parameters shift forward one; the second parameter becomes the first parameter, and so on.

Description

UNIQUE( ) returns *true* if the index tag specified by the <index position expN> parameter was created with the UNIQUE keyword or with SET UNIQUE ON; otherwise, it returns *false*.

The index must be referenced by number. If you do not specify an .MDX file to check, index numbering in the list of open indexes is complicated if you have open .NDX indexes or you have

open non-production .MDX files. For more information on index numbering, see SET INDEX. Either way, it is often easier to reference an index tag by name by using the TAGNO( ) function to get the corresponding position number.

If you do not specify an index tag, UNIQUE( ) checks the current master index tag and returns *false* if there is no master index.

If the specified .MDX file or index tag does not exist, UNIQUE( ) returns *false*.

OODML

No equivalent.

UNLOCK

Topic group Related topics

Releases all explicit locks.

Syntax

UNLOCK [ALL | IN <alias>]

ALL

Releases all explicit locks in all work areas in the current workset.

IN <alias>

Releases all explicit locks in the specified work area.

Description

Use UNLOCK to unlock file locks you obtained with FLOCK( ), or to unlock record locks you obtained with RLOCK( ) or LOCK( ). Issue UNLOCK at the same workstation as the one at which you issued the FLOCK( ), RLOCK( ), and LOCK( ) functions. UNLOCK can't release locks obtained through other workstations, and does not release automatic file and record locks.

When you set a relation from parent table to child tables with SET RELATION and then unlock the parent table or records in the parent table with UNLOCK, *dBASE Plus* also unlocks child tables or records. For more information on relating tables, see SET RELATION.

OODML

Use the Rowset object's *unlock*( ) method.

UPDATE

Topic group Related topics Example

Replaces data in the specified fields of the current table with data from another table.

Syntax

```
UPDATE ON <key field> FROM <alias>
REPLACE <field 1> WITH <exp 1>
  [, <field 2> WITH <exp 2>...]
[RANDOM]
[REINDEX]
```

<key field>

The key field that is common to both the current table and the table containing the updated information.

FROM <alias>

The work area that provides updates to the current table.

REPLACE <field 1>

The field in the current table to be updated with data from the table specified by FROM <alias>.

WITH <exp 1>

The expression to store in field <field 1>. Use the FROM table's alias name and the alias operator (that is, alias->field) to refer to field values in the FROM table.

[,<field n> WITH <exp n> ...]

Specifies additional fields to be updated.

RANDOM

Specifies the FROM table is neither indexed nor sorted. (The current table must be indexed on the key field common to both tables.)

REINDEX

Rebuilds open indexes after all records have been updated. Without REINDEX, *dBASE Plus* updates all open indexes after updating each record. When the current table has multiple open indexes or contains many records, UPDATE executes faster with the REINDEX option.

Description

The UPDATE command uses data from a specified table to replace field values in the current table. It makes the changes by matching records in the two files based on a single key field.

The current table must be indexed on the field in the key field. Unless the RANDOM option is used, the table in the specified work area should also be indexed or sorted on the same field. The key fields must have identical names in the two tables.

UPDATE works by traversing the FROM table, finding the matching record in the current table (the current table must be indexed or sorted so that the match can be found quickly), and executing the REPLACE clause. If there is no match for a record in the FROM table, it is ignored. If there are multiple records in the FROM table that match a single record in the current table, all the replacements will be applied. For a simple REPLACE clause, only the last one will appear to have taken effect.

SET EXACT affects the matching, so if you are using a language driver with both primary and secondary weights (not U.S. language drivers but most others) you should have SET EXACT ON.

OODML

Use the *update*( ) method of an UpdateSet object. Unlike the UPDATE command, the *update*( ) method updates all, rather than selected, fields in a row.

USE

Topic group Related topics Example

Opens the specified table and its associated index and memo files, if any.

Syntax

```
USE
[<filename1> [[TYPE] PARADOX | DBASE]
[IN <alias>]
[INDEX <filename2> [, <filename3> ...]]
```

```
[ORDER [TAG] <.ndx filename> |
  <tag name> [OF <.mdx filename>]]
[AGAIN]
[ALIAS <alias name>]
[AUTOMEM]
[EXCLUSIVE | SHARED]
[NOSAVE]
[NOUPDATE]]
```

<filename 1>

The table you want to open.

[TYPE] PARADOX | DBASE

Specifies the type of table you want to open, if <filename> does not include an extension. This option overrides the current SET DBTYPE setting. The TYPE keyword is included for readability only; it has no effect on the operation of the command.

IN <alias>

The work area in which to open the table. You can specify the work area that is being used by another table, in which case the other table is closed first.

INDEX <filename2> [, <filename3> ...]

Applicable to DBF indexes only. (Indexes on other table types are specified by the ORDER clause.) Opens up to 100 individual index files for the specified table, which can include single (.NDX) and multiple index file (.MDX) names and wildcards.

ORDER [TAG] <tag name>

Makes the <tag name> index file the master index.

If you don't include the ORDER clause and the first file name after INDEX is a single index .NDX file, the single index file is the master index. If you don't include ORDER and the first file name after INDEX is a multiple index .MDX file, the table is in natural order. If the table has a primary key index, it is used; otherwise the table is unordered.

OF <.mdx filename>

The .MDX file that includes <tag name>. Without OF <filename>, *dBASE Plus* searches for <tag name> in the table's production .MDX file, the .MDX file with the same root name as the table.

ORDER [TAG] <.ndx filename>

Makes the single index file, <.ndx filename>, the master index. The .NDX file must be specified in the INDEX clause. Use the name of the index without the file extension.

AGAIN

Opens a table and its related index files in the current or specified work area, leaving the table open in one or more other work areas. This keyword is superfluous and included for compatibility. *dBASE Plus* always opens tables with AGAIN.

ALIAS <alias name>

An alternate alias name to assign to the table.

AUTOMEM

Initializes a memory variable for each field of the specified table (not including memo, binary, or OLE types). The memory variables are assigned the same names and types as the fields.

EXCLUSIVE | SHARED

EXCLUSIVE opens the table so that no other users can open the table until you close it; SHARED allows other users access while the table is opened. This option overrides the current setting of SET EXCLUSIVE.

NOSAVE

Used to open a table as a temporary table. When you close a table opened with NOSAVE, it is erased along with its associated index and memo files. If you inadvertently open a table with the NOSAVE option, use COPY to save the data.

NOUPDATE

Prevents users from altering, deleting, or recalling any records in the table.

Description

The USE command opens an existing table and its associated files, including index and memo files. You need to open a table before you can access any data stored in the table.

USE with no options closes the open table and its associated files in the current work area. USE IN <alias>, with no other options, does the same in the specified work area. CLOSE TABLES closes tables in all work areas.

You can open a table in any work area. It is common practice to USE IN SELECT( ) to open the table in the first available work area. If a table is already opened in the specified work area, that table is closed before the specified table is opened.

USE...INDEX specifies index files that are opened and maintained for a particular table. For a DBF table, its production .MDX is automatically opened and does not need to be listed.

The ORDER option specifies the master index from the list of indexes opened with the INDEX option and the production .MDX index. USE...INDEX is identical to USE followed by SET INDEX. See the SET INDEX and SET ORDER commands for an explanation of the open index order and specifying a master index.

You can include .NDX as well as .MDX index file names with the INDEX option. If a table has an .NDX and an .MDX index file with the same name, *dBASE Plus* opens indexes listed in the .MDX index file. In that case, to open the .NDX file you would need to specify its full name, including its extension.

When opening a table, you can name the work area by including the ALIAS option in the USE command line. ALIAS names follow the same rules as file names. Aliases are used when referring to a table from another work area. If you do not specify an <alias name> the table name (without the extension) is used, unless that name is invalid, because:

- That alias name is already in use by another open table, perhaps because the table is already open in another work area, or
- The table name is not a valid alias name because it is a single letter from A to J or M, which are all reserved alias names, or some other reason.

If the table name is not a valid alias, a valid default alias is generated.

The AUTOMEM option creates blank automem variables for the table, as if the CLEAR AUTOMEM command was executed immediately after opening the table.

Use the NOSAVE option of USE to open a table as a temporary file. *dBASE Plus* automatically erases the table, along with its associated memo and index files, when you close the table.

To open a table read-only, which prevents intentional or accidental changes, use the NOUPDATE option.

OODML

Use a Query object with "SELECT * FROM <table>" as the *sql* property.

WORKAREA( )

Topic group Related topics

Returns a number representing the currently selected work area.

Syntax

WORKAREA( )

Description

The WORKAREA( ) function returns the number of the currently selected work area. Use WORKAREA( ) in a program to save the current work area number and then later restore that work area using the SELECT command.

Using the work area name returned by ALIAS( ) is generally preferred, but WORKAREA( ) will work better if there's a possibility that no table is in use in the current work area.

OODML

There is no concept of the "current" Query object. Use your usual object management techniques to manage Query objects.

ZAP

Topic group Related topics

Removes all records from the current table.

Syntax

ZAP

Description

ZAP is the fastest way to delete all records from a table. DELETE ALL, followed by PACK, also deletes all records from a table. Using ZAP requires a table be opened exclusively.

When SET SAFETY is ON and you issue ZAP, *dBASE Plus* displays a warning message asking you to confirm the operation before removing records.

OODML

Use the Database object's *emptyTable*( ) method.

Local SQL

Local SQL overview

Topic group Related topics

The Borland Database Engine (BDE) enables access to database tables through the industry-standard SQL language. Different table formats, for example InterBase¨ and Oracle, use different dialects of SQL. Local SQL (sometimes called "client-based SQL") is a subset of ANSI-92 SQL for accessing DB (Paradox) and DBF (dBASE) tables and fields (called "columns" in SQL).

Although it is called "local" SQL, the DB and DBF tables may reside on a remote network file server.

For information on the SQL dialect for other table formats, consult your SQL server documentation.

SQL statements are divided into two categories:

Data definition language
These statements are used for creating, altering, and dropping tables, and for creating and dropping indexes.

Data manipulation language
These statements are used for selecting, inserting, updating, and deleting table data.

In the examples, an SQL statement may be displayed on multiple lines for readability. But SQL is not line-oriented. When an SQL statement is specified in a string, as it is in a Query object's *sql* property, the entire SQL statement is specified in a single line. However, if you include a multi-line SQL statement in a program file, you must add semicolons to the end of each line (except the last) to act as line continuation characters; otherwise, the statement will not compile correctly.

SQL is not case-sensitive. The convention for SQL keywords is all uppercase, which is used in this series of Help topics. SQL statements in the rest of the *dBL Language Reference* may use either uppercase or lowercase.

Naming conventions

Topic group Related topics

This section describes the naming conventions for tables and columns in local SQL.

Tables

Local SQL supports full file and path specifications for table names. Table names with a path, spaces, or other special characters in their names must be enclosed in single or double quotation marks. You may use forward slashes instead of backslashes. For example,

```
SELECT * FROM PARTS.DB // Simple name with extension; no quotes required
SELECT * FROM "AIRCRAFT PARTS.DB" // Name has space; quotes needed
SELECT * FROM "C:\SAMPLE\PARTS.DB" // Filename with path
SELECT * FROM "C:/SAMPLE/PARTS.DB" // Forward slash instead of backslash
```

Local SQL also supports BDE aliases for table names. For example,

```
SELECT * FROM :IBAPPS:KBCAT
```

If you omit the file extension for a local table name, the table is assumed to be the table type specified the current setting of SET DBTYPE.

Finally, local SQL permits table names to duplicate SQL keywords as long as those table names are enclosed in single or double quotation marks. For example,

```
SELECT PASSID FROM "PASSWORD"
```

Columns

Local SQL supports multi-word column names and column names that duplicate SQL keywords as long as those column names are

Enclosed in single or double quotation marks
Prefaced with an SQL table name or table correlation name
For example, the following column name is two words:

```
SELECT E."Emp Id" FROM EMPLOYEE E
```

In the next example, the column name duplicates the SQL DATE keyword:

```
SELECT DATELOG."DATE" FROM DATELOG
```

Operators

Topic group Related topics

Local SQL supports the following operators:

Type	Operator	Type	Operator
Arithmetic	+ – * /	Logical	AND OR NOT
Comparison	< > = <> >= <= IS NULL IS NOT NULL	String concatenation	||

Note

The equality operator is a single equals sign; double equals are not allowed.

Reserved words

Topic group Related topics

The following is an alphabetical list of the 215 words reserved by local SQL:

ACTIVE	ADD	ALL	AFTER
ALTER	AND	ANY	AS
ASC	ASCENDING	AT	AUTO
AUTOINC	AVG	BASE_NAME	BEFORE
BEGIN	BETWEEN	BLOB	BOOLEAN
BOTH	BY	BYTES	CACHE
CAST	CHAR	CHARACTER	CHECK
CHECK_POINT_LENGTH	COLLATE	COLUMN	COMMIT
COMMITTED	COMPUTED	CONDITIONAL	CONSTRAINT
CONTAINING	COUNT	CREATE	CSTRING
CURRENT	CURSOR	DATABASE	DATE
DAY	DEBUG	DEC	DECIMAL
DECLARE	DEFAULT	DELETE	DESC
DESCENDING	DISTINCT	DO	DOMAIN
DOUBLE	DROP	ELSE	END
ENTRY_POINT	ESCAPE	EXCEPTION	EXECUTE
EXISTS	EXIT	EXTERNAL	EXTRACT
FILE	FILTER	FLOAT	FOR
FOREIGN	FROM	FULL	FUNCTION
GDSCODE	GENERATOR	GEN_ID	GRANT
GROUP	GROUP_COMMIT_WAIT_TIME	HAVING	HOUR
IF	IN	INT	INACTIVE

INDEX	INNER	INPUT_TYPE	INSERT
INTEGER	INTO	IS	ISOLATION
JOIN	KEY	LONG	LENGTH
LOGFILE	LOWER	LEADING	LEFT
LEVEL	LIKE	LOG_BUFFER_SIZE	MANUAL
MAX	MAXIMUM_SEGMENT	MERGE	MESSAGE
MIN	MINUTE	MODULE_NAME	MONEY
MONTH	NAMES	NATIONAL	NATURAL
NCHAR	NO	NOT	NULL
NUM_LOG_BUFFERS	NUMERIC	OF	ON
ONLY	OPTION	OR	ORDER
OUTER	OUTPUT_TYPE	OVERFLOW	PAGE_SIZE
PAGE	PAGES	PARAMETER	PASSWORD
PLAN	POSITION	POST_EVENT	PRECISION
PROCEDURE	PROTECTED	PRIMARY	PRIVILEGES
RAW_PARTITIONS	RDB$DB_KEY	READ	REAL
RECORD_VERSION	REFERENCES	RESERV	RESERVING
RETAIN	RETURNING_VALUES	RETURNS	REVOKE
RIGHT	ROLLBACK	SECOND	SEGMENT
SELECT	SET	SHARED	SHADOW
SCHEMA	SINGULAR	SIZE	SMALLINT
SNAPSHOT	SOME	SORT	SQLCODE
STABILITY	STARTING	STARTS	STATISTICS
SUB_TYPE	SUBSTRING	SUM	SUSPEND
TABLE	THEN	TIME	TIMESTAMP
TIMEZONE_HOUR	TIMEZONE_MINUTE	TO	TRAILING
TRANSACTION	TRIGGER	TRIM	UNCOMMITTED
UNION	UNIQUE	UPDATE	UPPER
USER	VALUE	VALUES	VARCHAR
VARIABLE	VARYING	VIEW	WAIT
WHEN	WHERE	WHILE	WITH
WORK	WRITE	YEAR	

Data definition

Topic group Related topics

Local SQL supports data definition language (DDL) for creating, altering, and dropping tables, and for creating and dropping indexes.

Local SQL does not permit the substitution of parameters for values in DDL statements.

The following DDL statements are supported:

CREATE TABLE

ALTER TABLE
DROP TABLE
CREATE INDEX
DROP INDEX

Data manipulation

Topic group Related topics

This section describes functions available to data manipulation language (DML) statements in local SQL. It covers

Parameter substitutions in DML statements
Aggregate functions
String functions
Date function
Updatable queries

With some restrictions, local SQL supports the following statements for data manipulation:

SELECT, for retrieving existing data
INSERT, for adding new data to a table
UPDATE, for modifying existing data
DELETE, for removing existing data from a table

Parameter substitutions in DML statements

Topic group Related topics

Parameters can be used in DML statements in place of values. Parameters must always be preceded by a colon (:). For example,

```
SELECT LAST_NAME, FIRST_NAME
FROM "CUSTOMER.DB"
WHERE LAST_NAME > :parm1 AND FIRST_NAME < :parm2
```

Assigning an SQL statement with parameters in a Query or StoredProc object automatically creates the corresponding elements in the object's params array. You then store values to substitute in that array.

Aggregate functions

Topic group Related topics

The following ANSI-standard SQL aggregate functions are available to local SQL for use with data retrieval:

SUM( ), for totaling all numeric values in a column
AVG( ), for averaging all non-NULL numeric values in a column
MIN( ), for determining the minimum value in a column
MAX( ), for determining the maximum value in a column
COUNT( ), for counting the number of values in a column that match specified criteria

Complex aggregate expressions are supported, such as

```
SUM( Field * 10 )
SUM( Field ) * 10
SUM( Field1 + Field2 )
```

String functions

Topic group Related topics Example

Local SQL supports the following ANSI-standard SQL string manipulation functions for retrieval, insertion, and updating:

UPPER( ), to force a string to uppercase:

```
UPPER(<expC>)
```

LOWER( ), to force a string to lowercase:

```
LOWER(<expC>)
```

TRIM( ), to remove repetitions of a specified character from the left, right, or both sides of a string:

```
TRIM( BOTH | LEADING | TRAILING
  <char> FROM <expC> )
```

SUBSTRING( ) to create a substring from a string:

```
SUBSTRING(<expC> FROM <start expN> FOR <length expN>)
```

You may use the LIKE predicate for pattern matching in the WHERE clause:

```
WHERE <expC> LIKE <pattern expC> [ESCAPE <char>]
```

In <pattern expC>, the % (percent) character stands for zero or more wildcard characters, and the _ (underscore) stands for a single wildcard character. To include either special character as an actual pattern character, specify an ESCAPE character and precede the wildcard character with that escape character.

Enclose literal strings in single quotes. To specify a single quote in a literal string, use double single quotes.

Date function

Topic group Related topics Example

Local SQL supports the EXTRACT( ) function for isolating a single numeric field from a date/time field on retrieval using the following syntax:

```
EXTRACT (<extract field> FROM <field name>)
```

where <extract_field> can be one of: YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND.

In local SQL, EXTRACT( ) does not support the TIMEZONE_HOUR or TIMEZONE_MINUTE clauses.

Updateable queries

Topic group Related topics

These restrictions apply to updates:

Linking fields cannot be updated
Index switching will cause an error

Restrictions on live queries

Topic group Related topics

Single-table queries are updatable provided that

- There are no JOINs, UNIONs, INTERSECTs, or MINUS operations.
- There is no DISTINCT key word in the SELECT. (This restriction may be relaxed if all the fields of a unique index are projected.)
- Everything in the SELECT clause is a simple column reference or a calculated field; no aggregation is allowed.
- The table referenced in the FROM clause is either an updatable base table or an updatable view.
- There is no GROUP BY or HAVING clause.
- There are no subqueries that reference the table in the FROM clause and no correlated subqueries.
- Any ORDER BY clause can be satisfied with an index (a simple single-field index for DBF tables).

Restrictions on live joins

Topic group Related topics

Live joins may be used only if:

- All joins are left-to-right outer joins.
- All join are equi-joins.
- All join conditions are satisfied by indexes.
- Output ordering is not defined.
- The query contains no elements listed above that would prevent single-table updatability.

Constraints

Topic group Related topics

You can constrain any updatable query by setting the Query object's *constrained* property to true before activating the query. This causes the query to behave more like an SQL-server-based query. New or modified rows that do not match the conditions of the query will disappear from the result set, although the data is saved.

Statements supported

Topic group Related topics

Local SQL supports the following DDL and DML statements:

ALTER TABLE

CREATE INDEX

CREATE TABLE

DELETE

DROP INDEX

DROP TABLE

INSERT

SELECT

FROM clause

WHERE clause

ORDER BY clause

GROUP BY clause

HAVING clause
UNION clause
UPDATE

ALTER TABLE

Topic group Related topics Example

Adds or drops (deletes) one or more columns (fields) from a table.

Syntax

```
ALTER TABLE table
 ADD <column name> <data type> |
 DROP <column name>
 [, ADD <column name> <data type> |
 DROP <column name> ...]
```

Description

Use ALTER TABLE to modify the structure of an existing table. ALTER TABLE with the ADD clause adds the column <*column name*> of the type <*data type*> to <*table name*>. Use the DROP clause to remove the existing column <*column name*> from <*table*>.

Warning!

Data stored in a dropped column is lost without warning, regardless of the SET SAFETY setting.

Adding an autoincrement field will pack the table. All records marked for deletion will be lost.

Multiple columns may be added and/or dropped in a single ALTER TABLE command.

Use ALTER TABLE as a means of modifying the structure of a table without using the Table Designer.

CREATE INDEX

Topic group Related topics Example

Creates a new index on a table.

Syntax

CREATE INDEX <*index name*> ON <*table name*> (<*column name*> [, <*column name*>...])

Description

Use CREATE INDEX to create a new index <*index name*>, in ascending order, based on the values in one or more columns <*column name*> of <*table name*>. Expressions cannot be used to create an index, only columns.

When working with DBF tables, the index can only be created for a single column. The new index is created as a new index tag in the production index. A production index is created if it does not exist. Using CREATE INDEX is the only way to create indexes for DBF tables in SQL.

CREATE INDEX can create only secondary indexes for Paradox tables. Primary Paradox indexes can be created only by specifying a PRIMARY KEY constraint when creating a new table with CREATE TABLE. The secondary indexes are created as case-insensitive and maintained, when possible.

CREATE INDEX is equivalent to the INDEX ON <*field list*> TAG <*tag name*> syntax in the dBL language.

CREATE TABLE

Topic group Related topics Example

Creates a new table.

Syntax

CREATE TABLE <table name> (<*column name*> <*data type*> [,<*column name*> <*data type*>...]
[, PRIMARY KEY(<*field name*>)])

Description

Create a FoxPro, Paradox or dBASE table using local SQL by specifying the file extension when naming the table:

DB for Paradox tables

DBF for FoxPro and dBASE tables

If you omit the file extension for a local table name, the table created is the table type specified in the Default Driver setting in the System page of the BDE Administrator.

CREATE TABLE has the following limitations:

Column definitions based on domains are not supported.

Constraints are limited to PRIMARY KEY. For DBF7 tables, only single-field primary keys are supported through the CREATE TABLE command. (Use the Table Designer or the Xbase INDEX command to create complex primary keys.) Primary keys are not supported for earlier versions of DBF.

At least one <*column name*> <*data type*> must be defined. The column definition list must be enclosed in parentheses.

CREATE TABLE is a alternate way of creating a table without using the Table Designer, the Database object's *copyTable( )* method, or an UpdateSet object.

Data type mappings for CREATE TABLE

Topic group Related topics

The following table lists SQL syntax for data types used with CREATE TABLE, and describes how those types are mapped to Paradox (DB) and dBASE (DBF) types by BDE:

SQL syntax	DB	DBF 7	DBF 5
SMALLINT	Short	Long	Numeric (6,10)
INTEGER	Long Integer	Long	Numeric (20,4)
DECIMAL(x,y)	BCD	Numeric (x,y)	N/A
NUMERIC(x,y)	Number	Numeric (x,y)	Numeric (x,y)
FLOAT(x,y)	Number	Double	Float (x,y)
CHARACTER(n)	Alpha	Character (n)	Character (n)
VARCHAR(n)	Alpha	Character (n)	Character (n)
DATE	Date	Date	Date
BOOLEAN	Logical	Logical	Logical
BLOB(n,1)	Memo	Memo	Memo

BLOB(n,2)	Binary	Binary	Binary
BLOB(n,3)	Formatted memo	N/A	N/A
BLOB(n,4)	OLE	OLE	OLE
BLOB(n,5)	Graphic	N/A	N/A
TIME	Time	N/A	N/A
TIMESTAMP	Timestamp	Timestamp	N/A
MONEY	Money	Numeric (20,4)	Numeric (20,4)
AUTOINC	Autoincrement	Autoincrement	N/A
BYTES(n)	Bytes	N/A	N/A

x = precision (default: specific to driver)

y = scale (default: 0)

n = length in bytes (default: 0)

1–5 = BLOB subtype (default: 1)

DELETE

Topic group Related topics Example

Deletes rows (records) from a table.

Syntax

DELETE FROM <*table name*> [WHERE <*search condition*>]

Description

Use DELETE to delete rows, or records, from <*table name*>. Without the WHERE clause, all the rows in the table are deleted. Use the WHERE clause to specify a <*search condition*>. Only records matching the <*search condition*> are deleted.

The Local SQL DELETE command is similar to the Xbase DELETE command; DELETE FROM with no WHERE clause is like the Xbase DELETE ALL.

In DBF tables, DELETE only **marks** rows as deleted; it does not remove them from the table. They may be recalled using the Xbase RECALL command. To remove the deleted records, use the Xbase PACK command. In Paradox tables, the rows are actually deleted, and are not recallable.

DROP INDEX

Topic group Related topics Example

Drops (deletes) an existing index from a table.

Syntax

DROP INDEX <table_name>.<index_name> | PRIMARY

Description

Use DROP INDEX to drop, or delete, the index <*index name*> from <*table name*>. For DBF tables <*index name*> must be the name of a tag in the production index.

The PRIMARY keyword is used to delete a primary Paradox index. For example, the following statement drops the primary index on EMPLOYEE.DB:

```
DROP INDEX "employee.db".PRIMARY
```

To drop any dBASE index, or to drop secondary Paradox indexes, provide the index name. For example, this statement drops a secondary index on a Paradox table:

```
DROP INDEX "employee.db".NAMEX
```

DROP TABLE

Topic group Related topics Example

Drops (deletes) a table.

Syntax

DROP TABLE <*table name*>

Description

Use DROP TABLE to delete the table <*table name*> from disk. The associated production index file and memo file, if any, are also deleted.

INSERT

Topic group Related topics Example

Adds new rows (records) to a table.

Syntax

INSERT INTO <*table name*>
[(<*column list*>)] VALUES (<*value list*>) |
SELECT <*command*>

Insertion from one table to another through a subquery is not allowed.

Description

Use INSERT to add rows, or records, to a table. There are two forms of this command. In the first form, you use <*value list*> to specify individual column values that are to be inserted for the new row. The values to be inserted must match in number, order, and type with the columns specified in <*column list*>, if <*column list*> is specified. Columns in the new row for which no value is given are left blank. If no <*column list*> is given, the order of the columns as they appear in the table is assumed. Without a <*column list*> a value must be provided for each column in the <*value list*>.

In the second form, the SELECT clause is executed just like a SELECT command. The row or rows returned by the SELECT are inserted into <*table name*>. The columns of the rows returned by the SELECT are matched up with the columns listed in <*column list*>. Therefore, the columns returned by SELECT must match in number, order, and type with the columns specified in <*column list*>, if <*column list*> is specified. If no <*column list*> is given, the number, order, and type of the columns returned by the SELECT must match the number, order, and type of the columns in <*table name*>.

SELECT

Topic group Related topics Example

Retrieves data from one or more tables.

Syntax

SELECT [DISTINCT] <column list>
FROM <table reference>
[WHERE <search condition>]
[ORDER BY <order list>]
[GROUP BY <group list>]
[HAVING <having condition>]
[UNION <select expr>]
[SAVE TO <table>]

Description

Use SELECT to retrieve data from a table or set of tables based on some criteria.

A SELECT that retrieves data from multiple tables is called a join.

<*column list*> is a comma-delimited list of columns in the table(s) you want to retrieve. The columns are retrieved in the order given in the list. If two or more tables used by SELECT use the same field names, distinguish the tables by using the table name and a dot (.). For example, if you're SELECTing from the CUSTOMER table and the PRODUCT table, and they both have a field called NAME, enter the fields as CUSTOMER.NAME and PRODUCT.NAME in <*column list*>. To retrieve all the columns from <*table list*>, use an asterisk (*) for <*column list*>. To make every retrieved row unique, add the DISTINCT keyword immediately after the keyword SELECT. Using the DESCENDING, or DESC, keyword in the ORDER BY clause will retrieve the columns in descending order (Z to A, 9 to 1, later dates to earlier dates).

For example, the following statement retrieves data from two columns:

```
SELECT PART_NO, PART_NAME
FROM PARTS
```

You may include calculated fields in the <column list>, optionally using the AS option to name them. For example:

```
SELECT LAST_NAME || ', ' || FIRST_NAME AS FULL_NAME, PHONE
FROM CUSTOMER
```

A SELECT statement that contains a join must have a WHERE clause in which at least one field from each table is involved in an equality check.

BDE Aliases

If a BDE Alias is open as either:

A database object

```
d = new Database()
d.databaseName := "BDEAliasName"
d.active := true )
```

or

Through the use of the OPEN DATABASE command

```
open database BDEAliasName )
```

the local SQL SELECT statement can be used to open a table:

```
select * from :BDEAliasName:TableName
```

A list of various syntax examples for the SELECT statement can be seen here: Examples

FROM clause

Topic group Related topics

The FROM clause specifies the table or tables from which to retrieve data. <table reference> can be a single table, a comma-delimited list of tables, or can be an inner or outer join as specified in the SQL-92 standard. For example, the following statement specifies a single table:

```
SELECT PART_NO FROM PARTS
```

The next statement specifies a left outer join for table_reference:

```
SELECT * FROM PARTS LEFT OUTER JOIN INVENTORY ;
ON PARTS.PART_NO = INVENTORY.PART_NO
```

WHERE clause

Topic group Related topics

The optional WHERE clause reduces the number of rows returned by a query to those that match the criteria specified in <search condition>. For example, the following statement retrieves only those rows with PART_NO greater than 543:

```
SELECT * FROM PARTS ;
WHERE PART_NO > 543
```

In addition to scalar comparison operators (=, <, > ...) additional predicates using IN, LIKE, ANY, ALL, and EXISTS are supported.

The IN predicate is followed by a list of values in parentheses. For example, the next statement retrieves only those rows where a part number matches an item in the IN predicate list:

```
SELECT * FROM PARTS ;
WHERE PART_NO IN (543, 544, 546, 547)
```

ORDER BY clause

Topic group Related topics

The ORDER BY clause specifies the order of retrieved rows, using the keywords ASC (the default) and DESC for ascending and descending, respectively. For example, the following query retrieves a list of all parts listed in alphabetical order by part name:

```
SELECT * FROM PARTS ;
ORDER BY PART_NAME ASC
```

The next query retrieves all part information ordered in descending numeric order by part number:

```
SELECT * FROM PARTS ;
ORDER BY PART_NO DESC
```

Calculated fields can be ordered by correlation name or ordinal position. For example, the following query orders rows by FULL_NAME, a calculated field:

```
SELECT LAST_NAME || ', ' || FIRST_NAME AS FULL_NAME, PHONE ;
FROM CUSTOMER ;
ORDER BY FULL_NAME
```

Projection of all grouping or ordering columns is not required.

GROUP BY clause

Topic group Related topics

The GROUP BY clause specifies how retrieved rows are grouped for aggregate functions. For example,

```
SELECT PART_NO, SUM(QUANTITY) AS PQTY ;
FROM PARTS ;
GROUP BY PART_NO
```

Aggregates in the SELECT clause must have a GROUP BY clause if a projected field is used, as shown in the example above.

HAVING clause

Topic group Related topics

The HAVING clause specifies conditions records must meet to be included in the return from a query. It is a conditional expression used in conjunction with the GROUP BY clause. Groups that do not meet the expression in the HAVING clause are omitted from the result set.

Subqueries are supported in the HAVING clause. A subquery works like a search condition to restrict the number of rows returned by the outer, or parent, query. See WHERE Clause.

In addition to scalar comparison operators (=, <, > ...) additional predicates using IN, LIKE, ANY, ALL, and EXISTS are supported.

UNION clause

Topic group Related topics

The UNION clause combines the results of two or more SELECT statements to produce a single table.

Heterogeneous joins

Topic group Related topics

Local SQL supports joins of tables in different database formats; such a join is called a heterogeneous join.

When you specify a table name after selecting a local alias,

For local tables, specify either the alias or the path.

For remote tables, specify the alias.

The following statement retrieves data from a Paradox table and a dBASE table:

```
SELECT DISTINCT C.CUST_NO, C.STATE, O.ORDER_NO
FROM CUSTOMER.DB C, ORDER.DBF O
WHERE C.CUST_NO = O.CUST_NO
```

You can also use BDE aliases in conjunction with table names.

SAVE TO clause

Topic group Related topics

The SAVE TO clause saves the data gathered by the SELECT into another table, instead of returning the result set. Use this option to copy part or all of a table into another table, or to save

the result of a join or aggregate to another table. For example, the following statement averages student scores by grade and stores the result to another table:

```
SELECT GRADE, AVG(SCORE);
FROM STUDENTS ;
GROUP BY GRADE ;
SAVE TO SCORES
```

UPDATE

Topic group Related topics Example

Adds or changes values in existing columns in existing rows of a table.

Syntax

UPDATE <*table name*>
SET <*column name*> = <*expression*> [, <*column name*> = <*expression*>...]
WHERE <*search condition*>

Description

Use UPDATE to update (change) values within existing columns in existing rows of a table. The column specified by <*column name*> is updated with the value of <*expression*> in all rows that match the <*search criteria*> of the WHERE clause. If the WHERE clause is omitted, the column is updated in all rows in the table. Multiple columns may be updated in a single UPDATE command. A given column of a table may only appear once to the left of an equal sign (=) in the SET clause.

Arrays

Array objects

Topic group

dBASE Plus supports a wide variety of array types:

- Arrays of contiguously numbered elements, in one or more dimensions. Elements are numbered from one. There are methods specifically for one- and two-dimensional arrays, which mimic a row of fields and a table of rows.
- Associative arrays, in which the elements are addressed by a key string instead of a number.
- Sparse arrays, which use non-contiguous numbers to refer to elements.

All arrays are objects, and use square brackets ([ ]) as indexing operators.

Array elements may contain any data type, including object references to other arrays. Therefore you can create nested arrays (multi-dimensional arrays of arrays with fixed length in each dimension), ragged arrays (nested arrays with variable lengths), arrays of associative arrays, and so on.

There are two array classes: Array and AssocArray. Sparse arrays can be created with any other object. In addition to creating properties by name, you can create numeric properties using the indexing operators. For example,

```
o = new Object()
o.title = "Summer"
o[ 2000 ] = "Sydney"
o[ 1996 ] = "Atlanta"
? o[ 1996 + 4 ] // Displays "Sydney"
```

Array functions

Topic group Related Topics

dBASE Plus supports a number of array functions, most of which have equivalent methods in the Array class. These functions are:

Function	Array class method
ACOPY( )	No equivalent
ADEL( )	delete( )
ADIR( )	dir( )
ADIREXT( )	dirExt( )
AELEMENT( )	element( )
AFIELDS( )	fields( )
AFILL( )	fill( )
AGROW( )	grow( )
AINS( )	insert( )
ALEN( )	For number of elements, check array's *size* property
ARESIZE( )	resize( )
ASCAN( )	scan( )
ASORT( )	sort( )
ASUBSCRIPT( )	subscript( )

Like the equivalent methods, these functions operate on one- and two-dimensional arrays only. ACOPY( ) and ALEN( ) are the only functions which have no direct equivalents.

The use of those functions is similar to the equivalent method. For a given method like:

```
aExample.fill( 0 ) // Fill array with zeros
```

the equivalent function uses the reference to the array as its first parameter and all other parameters (if any) following it:

```
afill( aExample, 0 )
```

The parameters following the array name in the function are identical to the parameters to the equivalent method, and the functions return the same values as the methods.

class Array

Topic group Related topics Example

An array of elements, in one or more dimensions.

Syntax

[<oRef> =] new Array([<dim1 expN> [,<dim2 expN>...]])

<oRef>

A variable or property in which to store a reference to the newly created Array object.

<dim1 expN> [,<dim2 expN> ...]

The size of the array in each specified dimension. If no dimensions are specified, the array is a one-dimensional array with zero elements.

Properties

The following tables list the properties and methods of the Array class. (No events are associated with this class.)

Property	Default	Description
baseClassName	ARRAY	Identifies the object as an instance of the Array class
className	(ARRAY)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
dimensions		The number of dimensions in the array
size	0	The number of elements in the array

Method	Parameters	Description
add( )	<exp>	Increases the size of a one-dimensional array by one and assigns the passed value to the new element.
delete( )	<position expN> [,1 | 2]	Deletes an element from a one-dimensional array, or deletes a row (1) or column (2) of elements from a two-dimensional array, without changing the size of the array.
dir( )	[<filespec expC>]	Stores in the array five characteristics of specified files: name, size, modified date, modified time, and file attribute(s). Returns the number of files whose characteristics are stored.
dirExt( )	[<filespec expC>]	Same as *dir*( ) method, but adds short (8.3) file name, create date, create time, and access date.
element( )	<row expN> [,<col expN>]	Returns the element number for the element at the specified row and column.
fields( )		Stores table structure information for the current table in the array
fill( )	<exp> , <start expN> [, <count expN>]	Stores a specified value into one or more elements of the array.
getFile( )	[<filename skeleton expC> [, <title expC> [, <suppress database expL>],[<file types list expC> | <group file name expC> (<file types list expC>)]]]	Displays a dialog box from which a user can select multiple files.
grow( )	1 | 2	When passed 1, adds a single element to a one-dimensional array or a row to a two-dimensional array; when passed 2, adds a column to the array.
insert( )	<element expN> [,1 | 2]	Inserts an element, row (1), or column (2) into an array without changing the size of the array (the last element, row, or column is lost).
resize( )	<rows expN> [, <cols expN> [, <retain values>]]	Increases or decreases the size of an array. First passed parameter indicates the new number of rows, the second parameter indicates the new number of columns. If the third parameter is zero, current values are relocated; if nonzero, they are retained in their old positions.
scan( )	<exp> , <start expN> [, <count expN>]	Searches an array for the specified expression; returns the element number of the first element that matches the expression, or zero if the search is unsuccessful.
sort( )	<start expN> [, <count expN> [, 0 | 1]]	Sorts the elements in a one-dimensional array or the rows in a two-dimensional array in ascending (0) or descending (1) order.
subscript( )	<element expN> 1 | 2	Returns the row (1) or column (2) subscript for the specified element number.

Description

An Array object is a standard array of elements, addressed by a contiguous range of numbers in one or more dimensions. The array can hold as many elements as memory allows. You can create arrays that contain more than two dimensions, but most dBL Array methods work only on one- or two-dimensional arrays. For a two-dimensional array, the first dimension is considered the row and the second dimension is the column. For example, the following statement creates an array with 3 rows and 4 columns:

```
a = new Array( 3, 4 )
```

There are two ways to refer to individual elements in an array; you can use either element subscripts or the element number. Element subscripts, one for each dimension, are values that represent the element's position in that dimension. For a two-dimensional array, they indicate the row and column in which an element is located. Element numbers indicate the sequential position of the element in the array, starting with the first element in the array and increasing in each dimension, with the last dimension first. For a two-dimensional array, the first element is in the first column of the first row, the second element is in the second column of the first row, and so on.

To determine the number of dimensions in an array, check its *dimensions* property (it's read-only). The array's *size* property reflects the number of elements in the array. To determine the number of rows or columns in a two-dimensional array, use the ALEN( ) function. There is no built-in way to determine the size of dimensions above two.

In an Array object, element numbering starts with one. You cannot create elements outside the defined range of elements or subscripts (although you could change the dimensions of the array if desired). For example, a 3-row, 4-column array has 12 elements, numbered 1 to 12. The first element's subscripts are [1,1] and the last element is [3,4].

Certain dBL methods require the element number, and others require the subscripts. If you are using one- or two-dimensional arrays, you can use *element*( ) to determine the element number if you know the subscripts, and *subscript*( ) to determine the subscripts if you know the element number.

Array elements may contain any data type, including object references to other arrays. Therefore you can create nested arrays (multi-dimensional arrays of arrays with fixed length in each dimension), ragged arrays (nested arrays with variable lengths), arrays of associative arrays, and so on.

With both nested and multi-dimensional arrays, you end up with multiple dimensions or levels of elements, but when you nest arrays, you create separate array objects, and the methods that are designed to work on the multiple dimensions of a single Array object will not work on the separate dimensions of the nested arrays.

In addition to creating an array with the NEW operator, you can create a populated one-dimensional array using the literal array syntax. For example, this statement

```
a1 = {"A", "B", "C"}
```

creates an Array object with three elements: "A", "B", and "C". You can nest literal arrays. For example, if this statement:

```
a2 = { {1, 2, 3}, a1 }
```

followed the first, you would then have a nested array.

To access a value in a nested array, use the index operators in series. Continuing the example, the third element in the first array would be accessed with:

```
x = a2[1][3] // 3
```

One-dimensional arrays are the only Array objects that are allowed to have zero elements. This is particularly useful for building arrays dynamically. To create a zero-element array, create a NEW Array with no parameters:

```
a0 = new Array( )
```

Then use the *add*( ) method to add elements to the array.

class AssocArray

Topic group Related topics Example

A one-dimensional associative array, in which the elements can be referenced by string.

Syntax

[<oRef> =] new AssocArray( )

<oRef>

A variable or property in which to store a reference to the newly created AssocArray object.

Properties

The following tables list the properties and methods of the AssocArray class. (No events are associated with this class.)

Property	Default	Description
baseClassName	ASSOCARRAY	Identifies the object as an instance of the AssocArray class
className	(ASSOCARRAY)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
firstKey		Character string assigned as the subscript of the first element of an associative array

Method	Parameters	Description
count( )		Returns the number of elements in the associative array
isKey( )	<key expC>	Returns *true* or *false* to indicate whether the character string is a key of the associative array
nextKey( )	<key expC>	Returns the associative array key following the passed key
removeAll( )		Deletes all elements from the associative array
removeKey( )	<key expC>	Deletes the specified element from the associative array

Description

In an associative array, elements are associated with arbitrary character strings, which act as key values. The keys may be of any length, and are case-sensitive. An AssocArray is a one-dimensional array.

New elements are created simply by assigning a value to a key. If the key does not exist, a new element is created. If the key already exists, then the old value is replaced. For example,

```
aTest = new AssocArray( )
aTest[ "alpha" ] = 1 // Create element with key "alpha" value 1
aTest[ "beta" ] = 2 // Create element with key "beta" value 2
aTest[ "alpha" ] = 3 // Change value of element "alpha" to 3
aTest[ "Beta" ] = 4 // Create element with key "Beta" value 4
```

The *isKey*( ) method will check if a given string is a key value in the associative array, and *removeKey*( ) will remove the element for a given key value from the array. *removeAll*( ) removes all the elements from the array.

The order of elements in an associative array is undefined. They are not necessarily sorted in the order they were added or sorted by their key values. You can think of an associative array as a bag of elements, and depending on what's in the bag, the order is different. But no matter what's in the associative array, you can use its *firstKey* property to get a key value, and use the *nextKey*( ) method to get all the other key values. The *count*( ) method will return the number of elements in the array so that you can call *nextKey*( ) as many times as needed.

ACOPY( )

Topic group Related topics

Copies elements from one array to another. Returns the number of elements copied.

Syntax

ACOPY(<source array>, <target array>
[, <starting element expN> [, <elements expN> [, <target element expN>]]])

<source array>

A reference to the array from which to copy elements.

<target array>

A reference to the array that elements are copied to.

<starting element expN>

The position of the element in <source array> from which ACOPY( ) starts copying. Without <starting element expN>, ACOPY( ) copies all the elements in <source array> to <target array>.

<elements expN>

The number of elements in <source array> to copy. Without <elements expN>, ACOPY( ) copies all the elements in <source array> from <starting element expN> to the end of the array. If you want to specify a value for <elements expN>, you must also specify a value for <starting element expN>.

<target element expN>

The position in <target array> to which ACOPY( ) starts copying. Without <target element expN>, ACOPY( ) copies elements to <target array> starting at the first position. If you want to specify a value for <target element expN>, you must also specify values for <starting element expN> and <elements expN>.

Description

ACOPY( ) copies elements from one array to another. The dimensions of the two array do not have to match; the elements are handled according to element number.

The target array must be big enough to contain all the elements being copied from the source array; otherwise no elements are copied and an error occurs.

add( )

Topic group Related topics Example

Adds an element to a one-dimensional array.

Syntax

<oRef>.*add*(<exp>)

<oRef>

A reference to the one-dimensional array to which you want to add the element.

<exp>

An expression of any type you want to assign to the new element.

Property of

Array

Description

Use *add*( ) to dynamically build a one-dimensional array.

add( ) adds a new element to a one-dimensional array and assigns <exp> to the new element.

You can create an empty one-dimensional array in a statement like:

```
a = new Array( ) // No parameters to Array class creates empty 1-D array
```

and add elements as needed.

ADEL( )

Topic group Related Topics

Deletes an element from a one-dimensional array, or deletes a row or column of elements from a two-dimensional array. Returns 1 if successful, an error if unsuccessful.

Syntax

ADEL(<array name>, <position expN> [, <row/column expN>])

<array name>

The name of the declared one- or two-dimensional array from which to delete data.

<position expN>

When <array name> is a one-dimensional array, <position expN> specifies the number of the element to delete.

When <array name> is a two-dimensional array, <position expN> specifies the number of the row or column whose elements you want to delete. The third argument (discussed in the next paragraph) specifies whether <position expN> is a row or a column.

<row/column expN>

Either 1 or 2. If you omit this argument or specify 1, a row is deleted from a two-dimensional array. If you specify 2, a column is deleted. *dBASE Plus* returns an error if you use <row/column expN> with a one-dimensional array.

Description

See the description for *delete( )*.

ADIR( )

Topic group Related Topics

Stores to a declared array five characteristics of specified files: name, size, date stamp, time stamp, and attribute(s). Returns the number of files whose characteristics are stored.

Syntax

ADIR(<array name>
[, <filename skeleton expC> [, <file attribute list expC>]])

<array name>

The name of the declared array of one or more dimensions to which to store the file information. ADIR( ) dynamically sizes <array name> so the number of rows in the array is equal to the number of files that match <file attribute expC>, and the number of columns is five.

<filename skeleton expC>

The file-name pattern (using wildcards) describing the files whose information to store to <array name>.

<file attribute list expC>

The letter or letters D, H, S, and/or V representing one or more file attributes.

Description

See the description for *dir( )*.

ADIREXT( )

Topic group Related Topics

On a Windows® 95/98/2000/NTsystem, stores to a declared array nine characteristics of specified files: name, size, date stamp, time stamp, attribute(s), alias, creation date, creation time, and last access date. Returns the number of files whose characteristics are stored.

Syntax

ADIREXT(<array name>
[, <filename skeleton expC> [, <file attribute list expC>]])

<array name>

The name of the declared array of one or more dimensions to which to store the file information. ADIREXT( ) dynamically sizes <array name> so the number of rows in the array is equal to the number of files that match <file attribute expC>, and the number of columns is nine.

<filename skeleton expC>

The file-name pattern (using wildcards) describing the files whose information to store to <array name>.

<file attribute list expC>

The letter or letters D, H, S, and/or V representing one or more file attributes. For more information, see ADIR( ).

Description

See the description for *dirExt( )*.

AELEMENT( )

Topic group Related Topics

Returns the number of a specified element in a one- or two-dimensional array.

Syntax

AELEMENT(<array name>, <subscript1 expN>
[, <subscript2 expN>])

<array name>

A declared one- or two-dimensional array.

<subscript1 expN>

The first subscript of the element. In a one-dimensional array, this is the same as the element number. In a two-dimensional array, this is the row.

<subscript2 expN>

When <array name> is a two-dimensional array, <subscript2 expN> specifies the second subscript, or column, of the element.

If <array name> is a two-dimensional array and you do not specify a value for <subscript2 expN>, *dBASE Plus* assumes the value 1. *dBASE Plus* returns an error if you use <subscript2 expN> with a one-dimensional array.

Description

See the description for *element( )*.

AFIELDS( )

Topic group Related Topics

Stores the current table's structural information to a declared array and returns the number of fields whose characteristics are stored.

Syntax

AFIELDS(<array name>)

<array name>

The name of a declared array of one or more dimensions.

Description

See the description for *fields( )*.

AFILL( )

Topic group Related Topics

Inserts a specified value into one or more locations in a declared array, and returns the number of elements inserted.

Syntax

AFILL(<array name>, <exp>
[, <start expN> [, <count expN>]])

<array name>

The name of a declared one- or two-dimensional array to fill with the specified value <exp>.

<exp>

An expression of character, date, logical, or numeric data type to insert in the specified array.

<start expN>

The element number at which to begin inserting <exp>. If you do not specify <start expN>, *dBASE Plus* begins at the first element in the array.

<count expN>

The number of elements in which to insert <exp>, starting at element <start expN>. If you do not specify <count expN>, *dBASE Plus* inserts <exp> from <start expN> to the last element in the array. If you want to specify a value for <count expN>, you must also specify a value for <start expN>.

If you do not specify <start expN> or <count expN>, *dBASE Plus* fills all elements in the array with <exp>.

Description

See the description for *fill( )*.

AGROW( )

Topic group Related Topics

Adds an element, row, or column to an array and returns a numeric value representing the number of added elements.

Syntax

AGROW (<array name>, <expN>)

<array name>

The name of a declared one- or two-dimensional array you want to add elements to.

< expN>

Either 1 or 2. When you specify 1, AGROW( ) adds a single element to a one-dimensional array or a row to a two-dimensional array. When you specify 2, AGROW( ) adds a column to the array.

Description

See the description for *grow( )*.

AINS( )

Topic group Related Topics

Inserts an element with the value *false* into a one-dimensional array, or inserts a row or column of elements with the value *false* into a two-dimensional array. Returns 1 if successful, an error if unsuccessful.

Syntax

AINS(<array name>, <position expN> [, <row/column expN>])

<array name>

The name of a declared one- or two-dimensional array in which to insert data.

<position expN>

When <array name> is a one-dimensional array, <position expN> specifies the number of the element in which to insert a value of *false*.

When <array name> is a two-dimensional array, <position expN> specifies the number of a row or column in which to insert *false* values. The third argument (discussed in the next paragraph) specifies whether <position expN> is a row or a column.

<row/column expN>

Either 1 or 2. If you omit this argument or specify 1, a row is inserted into a two-dimensional array. If you specify 2, a column is inserted. *dBASE Plus* returns an error if you use <row/column expN> with a one-dimensional array.

Description

See the description for *insert( )*.

ALEN( )

Topic group Related topics Example

Returns the number of elements, rows, or columns of an array.

Syntax

ALEN(<array> [, <expN>])

<array>

A reference to a one- or two-dimensional array.

<expN>

The number 0, 1, or 2, indicating which array information to return: elements, rows, or columns. The following table describes what ALEN( ) returns for different <expN> values:

If <expN> is...	ALEN( ) returns...
not supplied	Number of elements in the array
0	Number of elements in the array
1	For a one-dimensional array, the number of elements
	For a two-dimensional array, the number of rows (the first subscript of the array)
2	For a one-dimensional array, 0 (zero)
	For a two-dimensional array, the number of columns (the second subscript of the array)
any other value	0 (zero)

Property of

Array

Description

Use ALEN( ) to determine the dimensions of an array—either the number of elements it contains, or the number of rows or columns it contains.

The number of elements in an array (with any number of dimensions) is also reflected in the array's *size* property.

If you need to determine both the number of rows and the number of columns a two-dimensional array contains, call ALEN( ) twice, once with a value of 1 for <expN> and once with a value of 2 for <expN>. For example, the following determines the number of rows and columns contained in aExample:

```
nRows = alen( aExample, 1 )
nCols = alen( aExample, 2 )
```

ARESIZE( )

Topic group Related Topics

Increases or decreases the size of an array according to the specified dimensions and returns a numeric value representing the number of elements in the modified array.

Syntax

ARESIZE(<array name>, <new rows expN>
[,<new cols expN> [, <retain values expN>]])

<array name>

The name of a declared one- or two-dimensional array whose size you want to increase or decrease.

<new rows expN>

The number of rows the resized array should have. <new rows expN> must always be a positive, nonzero value.

<new cols expN>

The number of columns the resized array should have. <new cols expN> must always be zero or a positive value. If you omit this option, ARESIZE( ) changes the number of rows in the array and leaves the number of columns the same.

<retain values expN>

Determines what happens to the values of the array when rows are added or removed. If you want to specify a value for <retain values expN>, you must also specify a value for <new cols expN>.

Description

See the description for *resize( )*.

ASCAN( )

Topic group Related Topics

Searches an array for an expression. Returns the number of the first element that matches the expression if the search is successful, or zero if the search is unsuccessful.

Syntax

ASCAN(<array name>, <exp>
[, <starting element expN> [, <elements expN>]])

<array name>

A declared one- or two-dimensional array.

<exp>

The expression to search for in <array name>.

<starting element expN>

The element number in <array name> at which to start searching. Without <starting element expN>, ASCAN( ) starts searching at the first element.

<elements expN>

The number of elements in <array name> that ASCAN( ) searches. Without <elements expN>, ASCAN( ) searches <array name> from <starting element expN> to the end of the array. If you want to specify a value for <elements expN>, you must also specify a value for <starting element expN>.

Description

See the description for *scan( )*.

ASORT( )

Topic group Related Topics

Sorts the elements in a one-dimensional array or the rows in a two-dimensional array, returning 1 if successful or an error if unsuccessful.

Syntax

ASORT(<array name>
[, <starting element expN> [,<elements to sort expN>
[, <sort order expN>]]])

<array name>

A declared one- or two-dimensional array.

<starting element expN>

In a one-dimensional array, the number of the element in <array name> at which to start sorting. In a two-dimensional array, the number (subscript) of the column on which to sort. Without <starting element expN>, ASORT( ) starts sorting at the first element or column in the array.

<elements to sort expN>

In a one-dimensional array, the number of elements to sort. In a two-dimensional array, the number of rows to sort. Without <elements to sort expN>, ASORT( ) sorts the rows starting at the row containing element <starting element expN> to the last row. If you want to specify a value for <elements to sort expN>, you must also specify a value for <starting element expN>.

<sort order expN>

The sort order:

0 specifies ascending order (the default)
1 specifies descending order

If you want to specify a value for <sort order expN>, you must also specify values for <elements to sort expN> and <starting element expN>.

Description

See the description for *sort( )*.

ASUBSCRIPT( )

Topic group Related Topics

Returns the row number or the column number of a specified element in an array.

Syntax

ASUBSCRIPT(<array name>, <element expN>, <row/column expN>)

<array name>

A declared one- or two-dimensional array.

<element expN>

The element number.

<row/column expN>

A number, either 1 or 2, that determines whether you want to return the row or column subscript of an array. If <row/column expN> is 1, ASUBSCRIPT( ) returns the number of the row subscript. If <row/column expN> is 2, ASUBSCRIPT( ) returns the number of the column subscript.

If <array name> is a one-dimensional array, *dBASE Plus* returns an error if <row/column expN> is a value other than 1.

Description

See the description for *subscript( )*.

count( )

Topic group Related topics Example

Returns the number of elements in an associative array.

Property of

AssocArray

Description

Use *count*( ) to determine the number of elements in an associative array.

Because associative arrays use arbitrary strings as keys and change size dynamically, you need to get the number of elements in an associative array if you want to loop through its elements.

DECLARE

Topic group Related Topics Example

Defines one or more fixed arrays.

Syntax

DECLARE <array name 1>"["<expN list 1>"]"
[,<array name 2>"["<expN list 2>"]" ...]

Brackets ([ ]) in quotation marks are required syntax components.

<array name 1>[,<array name 2> ...]

The memory variable(s) that are the name(s) of the array(s).

"["<expN list 1>"]"[,..."["<expN list 2>"]"][,...]

Numeric expressions (from 1 to 254 inclusive). The number of expressions you specify determines the number of dimensions of the array. Each one of the expressions specifies how many values (data elements) that dimension has. For example, if [<expN list 1>] is [3,4], *dBASE Plus* defines a two-dimensional array with three rows and four columns.

Description

Use DECLARE to define an array of a specified size as a memory variable. Array elements can be of any data type. (An array element can also specify the name of another array.) A single array can contain multiple data types. When you use DECLARE, all array elements are initialized to a logical data type with a value of .F.

The array can hold as many elements as memory allows. You can create arrays that contain more than two dimensions, but most dBL array functions work only on one- or two-dimensional arrays.

There are two ways to refer to individual elements in an array; you can use either the element subscripts or the element number. Element subscripts indicate the row and column in which an element is located. Element numbers indicate the sequential position of the element in the array, starting at the first row and first column of the array. To determine the number of elements, rows, or columns in an array, use ALEN( ).

Certain dBL functions require the element number, and others require the subscripts. If you are using one- or two-dimensional arrays, you can use AELEMENT( ) to determine the element number if you know the subscripts, and ASUBSCRIPT( ) to determine the subscripts if you know the element number.

After you create an array, you can place values in cells of the array using STORE, or you can use =. You can also use AFILL( ) to place the same value in a range of cells in the array. To add or delete elements from an array, use ADEL( ) and AINS( ). To resize an array, or make a one-dimensional array two-dimensional, use AGROW( ) or ARESIZE( ).

You can pass array elements as parameters, and you can pass a complete array as a parameter to a program or procedure by specifying the array name without a subscript.

delete( )

Topic group Related topics Example

Deletes an element from a one-dimensional array, or deletes a row or column of elements from a two-dimensional array. Returns 1 if successful; generates an error if unsuccessful. The remaining elements move forward to replace the deleted element(s); the dimensions of the array do not change.

Syntax

<oRef>.*delete*(<position expN> [, <row/column expN>])

<oRef>

A reference to the one- or two-dimensional array from which you want to delete data.

<position expN>

When the array is a one-dimensional array, <position expN> specifies the number of the element to delete.

When the array is a two-dimensional array, <position expN> specifies the number of the row or column whose elements you want to delete. The second argument (discussed in the next paragraph) specifies whether <position expN> is a row or a column.

<row/column expN>

Either 1 or 2. If you omit this argument or specify 1, a row is deleted from a two-dimensional array. If you specify 2, a column is deleted. *dBASE Plus* generates an error if you use <row/column expN> with a one-dimensional array.

Property of

Array

Description

Use *delete*( ) to delete selected elements from an array without changing the size of the array. *delete*( ) does the following:

- Deletes an element from a one-dimensional array, or deletes a row or column from a two-dimensional array
- Moves all remaining elements toward the beginning of the array (up if a row is deleted, to the left if an element or column is deleted)
- Inserts *false* values in the last position(s)

Adjust the array's *size* property or use *resize*( ) to make the array smaller after you *delete*( ) if you want the net effect of removing elements.

One-dimensional arrays

When you issue *delete*( ) for a one-dimensional array, the element in the specified position is deleted, and the remaining elements move one position toward the beginning of the array. The logical value *false* is stored to the element in the last position.

For example, if you define a one-dimensional array with

```
aAlpha = {"A", "B", "C"}
```

the resulting array has one row and can be illustrated as follows:

```
A B C
```

Issuing aAlpha.*delete*(2) deletes element number 2 whose value is "B," moves the value in aAlpha[3] to aAlpha[2], and stores *false* to aAlpha[3] so that the array now contains these values:

```
A C false
```

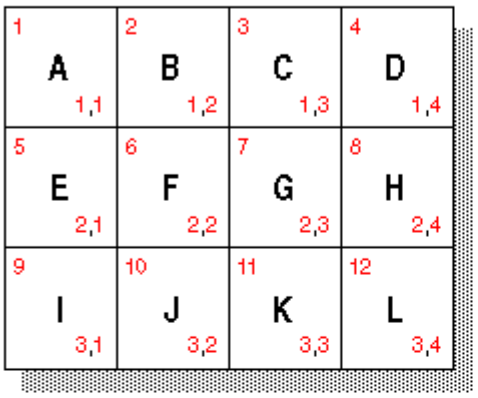
Two-dimensional arrays

When you issue *delete*( ) for a two-dimensional array, the elements in the specified row or column are deleted, and the elements in the remaining rows or columns move one position toward the beginning of the array. The logical value *false* is stored to the elements in the last row or column.

For example, suppose you define a two-dimensional array and store letters to the array. The following illustration shows how the array is changed by aAlpha.*delete*(2,2).

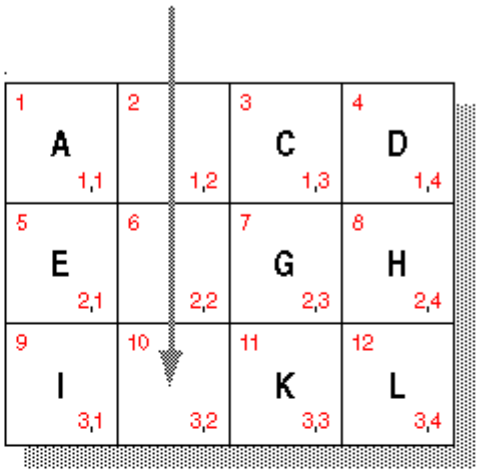
1 Original array created as:

```
aAlpha = new Array(3,4)
aAlpha[1,1] = "A"
aAlpha[1,2] = "B"
   ⋮
aAlpha[3,4] = "L"
```

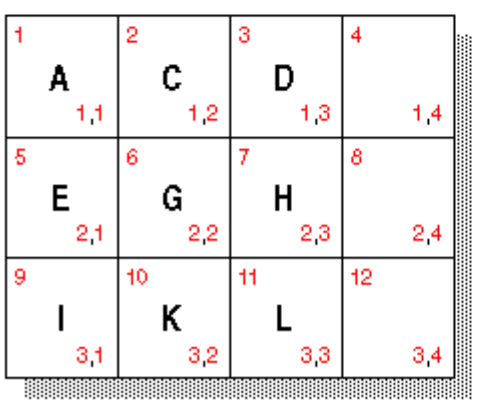


Initial contents of the array aAlpha

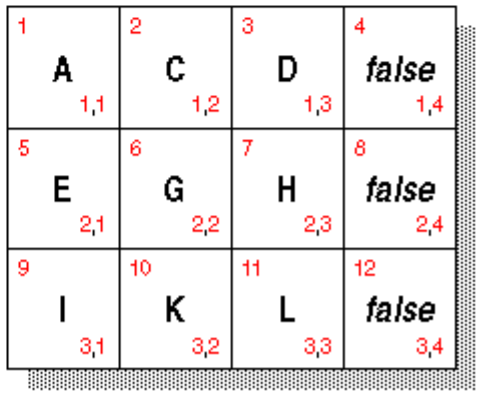
2 aAlpha.*delete*(2,2) deletes the elements in the second column...



3 Shifts the elements in the remaining columns towards the beginning of the array...



4 And inserts logical *false* values as elements in the last column, resulting in this array:



Contents of the array after issuing aAlpha.delete(2,2)

dimensions

Topic group Related topics Example

The number of dimensions in an Array object.

Property of

Array

Description

dimensions indicates the number of dimensions in an Array object. It is a read-only property.

You can use the *resize*( ) method to change the number of dimensions to one or two, but for more than two you would have to create a new array.

If the array has one or two dimensions, you can use the ALEN( ) function to determine the size of each dimension. There is no built-in way to determine dimension sizes for arrays with more than two dimensions.

dir( )

Topic group Related topics Example

Fills the array with five characteristics of specified files: name, size, modified date, modified time, and file attribute(s). Returns the number of files whose characteristics are stored.

Syntax

<oRef>.*dir*([<filename skeleton expC> [, <DOS file attribute list expC>]])

<oRef>

A reference to the array in which you want to store the file information. *dir*( ) will automatically redimension or increase the size of the array to accommodate the file information, if necessary.

<filename skeleton expC>

The file-name pattern (using wildcards) describing the files whose information you want to store to <oRef>.

<file attribute list expC>

The letter or letters D, H, S, and/or V representing one or more file attributes.

If you want to specify a value for <file attribute expC>, you must also specify a value or "*.*" for <filename skeleton expC>.

The meaning of each attribute is as follows:

Character	Meaning
D	Directories
H	Hidden files
S	System files
V	Volume label

If you supply more than one letter for <file attribute expC>, include all the letters between one set of quotation marks, for example, aFiles.*dir*("*.*", "HS").

Property of

Array

Description

Use *dir*( ) to store information about files to an array, which is dynamically resized so all returned information fits in the array. The resulting array is always a two-dimensional array, unless there are no files, in which case the array is not modified.

Without <filename skeleton expC>, *dir*( ) stores information about all files in the current directory, unless they are hidden or system files. For example, if you want to return information only on DBF tables, use "*.DBF" as <filename skeleton expC>.

If you want to include directories, hidden files, or system files in the array, use <file attribute expC>. When D, H, or S is included in <file attribute expC>, all directories, hidden files, and/or system files (respectively) that match <filename skeleton expC> are added to the array.

When V is included in <file attribute expC>, *dir*( ) ignores <filename skeleton expC> as well as other characters in the attribute list, and stores the volume label to the first element of the array.

dir( ) stores the following information for each file in each row of the array. The data type for each is shown in parentheses:

Column 1	Column 2	Column 3	Column 4	Column 5
File name (character)	Size (numeric)	Modified date (date)	Modified time (character)	File attribute(s) (character)

The last column (file attribute) can contain one or more of the following file attributes, in the order shown:

Attribute	Meaning
R	Read-only file
A	Archive file (modified since it was last backed up)
S	System file
H	Hidden file
D	Directory

If the file has the attribute, the letter code is in the column. Otherwise, there is a period. For example, a file with none of the attributes would have the following string in column 5:

```
.....
```

A read-only, hidden file would have the following string in column 5:

```
R..H.
```

Use *dirExt*( ) to get extended Windows® 95/98/2000/NT/ME file information.

dirExt( )

Topic group Related topics Example

dirExt( ) is an extended version of the *dir*( ) method. It fills the array with nine characteristics of specified files: name, size, modified date, modified time, file attribute(s), short (8.3) file name, create date, create time, and access date. Returns the number of files whose characteristics are stored.

Syntax

<oRef>.*dirExt*([<filename skeleton expC> [, <file attribute list expC>]])

<oRef>

A reference to the array in which you want to store the file information. *dirExt*( ) will automatically redimension or increase the size of the array to accommodate the file information, if necessary.

<filename skeleton expC>

The file-name pattern (using wildcards) describing the files whose information you want to store to <oRef>.

<file attribute list expC>

The letter or letters D, H, S, and/or V representing one or more file attributes.

If you want to specify a value for <file attribute expC>, you must also specify a value or "*.*" for <filename skeleton expC>.

The meaning of each attribute is as follows:

Character	Meaning
D	Directories
H	Hidden files
S	System files
V	Volume label

If you supply more than one letter for <file attribute expC>, include all the letters between one set of quotation marks, for example, aFiles.*dirExt*("*.*", "HS").

Property of

Array

Description

Use *dirExt*( ) to store information about files to an array, which is dynamically resized so all returned information fits in the array. The resulting array is always a two-dimensional array, unless there are no files, in which case the array is not modified.

Without <filename skeleton expC>, *dirExt*( ) stores information about all files in the current directory, unless they are hidden or system files. For example, if you want to return information only on DBF tables, use "*.DBF" as <filename skeleton expC>.

If you want to include directories, hidden files, or system files in the array, use <file attribute expC>. When D, H, or S is included in <file attribute expC>, all directories, hidden files, and/or system files (respectively) that match <filename skeleton expC> are added to the array.

When V is included in <file attribute expC>, *dirExt*( ) ignores <filename skeleton expC> as well as other characters in the attribute list, and stores the volume label to the first element of the array.

dirExt( ) stores the following information for each file in each row of the array. The data type for each is shown in parentheses:

Column 1	Column 2	Column 3	Column 4	Column 5
File name (character)	Size (numeric)	Modified date (date)	Modified time (character)	File attribute(s) (character)

Column 6	Column 7	Column 8	Column 9
Short (8.3) file name (character)	Create date (date)	Create time (character)	Access date (date)

Column 5 (file attribute) can contain one or more of the following file attributes, in the order shown:

Attribute	Meaning
R	Read-only file
A	Archive file (modified since it was last backed up)
S	System file

H	Hidden file
D	Directory

If the file has the attribute, the letter code is in the column. Otherwise, there is a period. For example, a file with none of the attributes would have the following string in column 5:

```
.....
```

A read-only, hidden file would have the following string in column 5:

```
R..H.
```

Use *dir*( ) to get basic file information only.

element( )

Topic group Related topics Example

Returns the number of a specified element in a one- or two-dimensional array.

Syntax

<oRef>.*element*(<subscript1 expN> [, <subscript2 expN>])

<oRef>

A reference to a one- or two-dimensional array.

<subscript1 expN>

The first subscript of the element. In a one-dimensional array, this is the same as the element number. In a two-dimensional array, this is the row.

<subscript2 expN>

When <oRef> is a two-dimensional array, <subscript2 expN> specifies the second subscript, or column, of the element.

If <oRef> is a two-dimensional array and you do not specify a value for <subscript2 expN>, *dBASE Plus* assumes the value 1, the first column in the row. *dBASE Plus* generates an error if you use <subscript2 expN> with a one-dimensional array.

Property of

Array

Description

Use *element*( ) when you know the subscripts of an element in a two-dimensional array and need the element number for use with another method, such as *fill*( ) or *scan*( ).

In one-dimensional arrays, the number of an element is the same as its subscript, so there is no need to use *element*( ). For example, if aOne is a one-dimensional array, aOne.*element*(3) returns 3, aOne.*element*(5) returns 5, and so on.

element( ) is the inverse of *subscript*( ), which returns an element's row or column subscript number when you specify the element number.

fields( )

Topic group Related topics

Fills the array with the current table's structural information. Returns the number of fields whose characteristics are stored.

Syntax

<oRef>.*fields*( )

<oRef>

A reference to the array in which you want to store the field information. *fields*( ) will automatically redimension or increase the size of the array to accommodate the field information, if necessary.

Property of

Array

Description

Use *fields*( ) to store information about the structure of the current table to an array, which is dynamically resized so all returned information fits in the array. The resulting array is always a two-dimensional array, unless there are is no table open in the current work area, in which case the array is not modified.

fields( ) stores the following information for each field in each row of the array. The data type for each is shown in parentheses:

Column 1	Column 2	Column 3	Column 4
Field name (character)	Field type (character)	Field length (numeric)	Decimal places (numeric)

dBASE Plus uses the following codes for field types (some codes are used for more than one field type):

Code	Field type
B	Binary
C	Character, Alphanumeric
D	Date, Timestamp
F	Float, Double
G	General, OLE
L	Logical, Boolean
M	Memo
N	Numeric

fields( ) stores the same information into an array that COPY STRUCTURE EXTENDED stores into a table, except *fields*( ) doesn't create a column containing FIELD_IDX information.

fill( )

Topic group Related topics Example

Stores a specified value into one or more locations in an array, and returns the number of elements stored.

Syntax

<oRef>.*fill*(<exp> [, <start expN> [, <count expN>]])

<oRef>

A reference to a one- or two-dimensional array you want to fill with the specified value <exp>.

<exp>

An expression you want to store in the specified array.

<start expN>

The element number at which you want to begin storing <exp>. If you do not specify <start expN>, *dBASE Plus* begins at the first element in the array.

<count expN>

The number of elements in which you want to store <exp>, starting at element <start expN>. If you do not specify <count expN>, *dBASE Plus* stores <exp> from <start expN> to the last element in the array. If you want to specify a value for <count expN>, you must also specify a value for <start expN>.

If you do not specify <start expN> or <count expN>, *dBASE Plus* fills all elements in the array with <exp>.

Property of

Array

Description

Use *fill*( ) to store a value into all or some elements of an array. For example, if you are going to use elements of an array to calculate totals, you can use *fill*( ) to initialize all values in the array to 0.

fill( ) stores values into the array sequentially. Starting at the first element in the array or at the element specified by <start expN>, *fill*( ) stores the value in each element in a row, then moves to the first element in the next row, continuing to store values until the array is filled or until it has inserted <count expN> elements. *fill*( ) overwrites any existing data in the array.

If you know an element's subscripts, you can use *element*( ) to determine its element number for use as <start expN>.

firstKey

Topic group Related topics Example

Returns the character string key for the first element of an associative array.

Property of

AssocArray

Description

Use *firstKey* when you want to loop through the elements in an associative array. Once you have gotten the key value for the first element with *firstKey*, use *nextKey*( ) to get the key values for the rest of the elements.

Note

The order of elements in an associative array is undefined. They are not necessarily stored in the order in which you add them, or sorted by their key values. You can't assume that the value returned by *firstKey* will be consistent, or that it will return the first item you added.

For an empty associative array, *firstKey* is the logical value *false*. Because *false* is a different data type than valid key values (which are character strings), it requires extra code to look for *false* to see if the array is empty. It's easier to get the number of elements in the array with *count*( ) and see if it's greater than zero.

getFile( )

Topic group Related topics

Displays a dialog box from which a user can select multiple files.

Syntax

<oRef>.*getFile*([<filename skeleton expC> [, <title expC> [, <suppress database expL>], [<file types list expC> | <group file name expC> (<file types list expC>)]]])

<oRef>

A reference to the array in which the selected filenames, or database aliases, will be stored.

<filename skeleton expC>

A character string that specifies which files are to be displayed in the *getFile*( ) dialog. It may contain a valid path followed by a filename skeleton.

If a path was specified, it is used to set the initial path from which *getFile*( ) displays files.

If a path was not specified, the path from any previously run *getFile*( ) method, GETFILE( ) function, or PUTFILE( ) function will be used as the new initial path. If no previous path exists, the *getFile*( ) method uses the current dBASE directory - the path returned by the SET("DIRECTORY") function - as the initial path.

If no filename skeleton is specified, "*.*" is assumed and the *getFile*( ) method displays all files in the initial path described above.

<title expC>

A title displayed in the top of the dialog box. Without <title expC>, the *getFile*( ) methods' dialog box displays the default title. If you want to specify a value for <title expC>, you must also specify a value, or empty string (""), for <filename skeleton expC>.

<suppress database expL>

Whether to suppress the combobox from which you can choose a database. The default is *true* (the Database combobox is not displayed). If you want to specify a value for <suppress database expL>, you must also specify a value, or empty string (""), for <filename skeleton expC> and <title expC>.

<file types list expC>

A character expression containing zero, or more, file types to be displayed in the "Files of Type" combobox. If this parameter is not specified, the following file types will be loaded into the “Files of Type” combobox:

Projects (*.prj)
Forms (*.wfm;*.wfo)
Custom Forms (*.cfm;*.cfo)
Menus (*.mnu;*.mno)
Popup (*.pop;*.poo)
Reports (*.rep;*.reo)
Custom Reports (*.crp;*.cro)
Labels (*.lab;*.lao)
Programs (*.prg;*.pro)
Tables (*.dbf;*.db)
SQL (*.sql)
Data Modules (*.dmd;*.dmo)
Custom Data Modules (*.cdm;*.cdo)
Images (*.bmp;*.ico;*.gif;*.jpg;*.jpeg;*.pje;*.xbm)
Custom Components (*.cc;*.co)
Include (*.h)
Executable (*.exe)

Sound (*.wav)
Text (*.txt)
All (*.*)

<file type group name expC>

A character expression denoting a custom or built in file type goup name to use in the dialog.

For a custom file type group name

Specify the group name followed by a list of one or more file types within parenthesis

For example, "Web Images (*.jpg,*.jpeg,*.bmp)"

For a built-in file type group name

Specify the group name followed by a set of empty parenthesis

For example, "Images ()"

dBASE will detect the left and right parenthesis and search for a matching built-in group name. If found, the list of file types associated with the built-in group will be added to the string that is added to the File Types combobox in the GetFile() or PutFile() dialog.

File Types between parenthesis must be separated by either a comma or a semi-colon and can be specified in any of the following formats:

<ext> or .<ext> or *.<ext>

where <ext> means a file extension such as jpg or txt. If a File Type List string is specified that contains unbalanced parenthesis or right parenthesis before left parenthesis, a new error will be triggered with the message:

"Unbalanced parenthesis in file type list"

Examples using file group name:

```
a.getFile("","Title",true,"Web Images (*.jpg,*.png,*.gif,*.bmp)")
```

or if you want to use one of the built in file type groups:

```
a.getFile("","Title",true,"FILE ()")
```

If an empty string is specified, "All (*.*)" will be loaded into the Files of Type combobox.

If one or more file types are specified, dBASE will check each file type specified against an internal table.

If a match is found, a descriptive character string will be loaded into the "Files of Type" combobox.

If a matching file type is not found, a descriptive string will be built, using the specified file type, in the form

`"<File Type> files (*.<File Type>)"`

and will be loaded into the "Files of Type" combobox.

When the expression contains more than one file type, they must be separated by either commas or semicolons.

File types may be specified with, or without, a leading period.

The special extension ".*" may be included in the expression to specify that "All (*.*)" be included in the Files of Type combobox.

File types will be listed in the Files of Type combobox, in the order specified in the expression.

Note: In Visual dBASE 5.x, the GETFILE( ) and PUTFILE( ) functions accepted a logical value as a parameter in the same position as the new <file types list expC> parameter. This logical value has no function in dBASE Plus. However, for backward compatibility, dBASE Plus will ignore a logical value if passed in place of the <file types list expC>.

Examples of <file types list expC> syntax.can be viewed here

Description

Use the *getFile*( ) method to display a dialog box from which the user can choose, or enter, one or more existing file names. Any elements already in the array will be released.

Pressing the dialog's "Open" button closes the dialog and adds the selected files to the array.

The resulting array will contain a single column. Each element of the array will contain a single file path and name or, if selected from a database, the database alias followed by the table name.

The *getFile*( ) method returns the number of files selected, or zero if none are selected or the dialog is cancelled.

Example

```
a = new array
if ( a.getFile("*.*", "Choose Files", true) > 0 )
  // Do something with the chosen files
endif
```

File list size

During file selection, selected file names are stored in a buffer with quotes around each file name and a space between them. The maximum number of files that can be selected is limited by the size of this buffer due to the Windows common file dialog requirement that the buffer be preallocated before opening the dialog.

When the.*getFile*( ) method is executed, it attempts to allocate a buffer with the sizes shown below. However, if the memory allocation is unsuccessful it cuts the requested size in half and tries again. It continues looping in this fashion until a successful allocation occurs, or the requested size becomes equal to zero. Should the requested file size become equal to zero, multifile selection is disabled and only a single file selection is allowed.

The maximum buffer sizes are:

Win 9x:

512*260 = 133120 filename characters

Win NT, 2000, XP:

4030*260 characters (approximately 1 megabyte of filename characters).

grow( )

Topic group Related topics Example

Adds an element, row, or column to an array and returns the number of added elements.

Syntax

<oRef>.*grow*(<expN>)

<oRef>

A reference to a one- or two-dimensional array you want to add elements to.

<expN>

Either 1 or 2. When you specify 1, *grow*( ) adds a single element to a one-dimensional array or a row to a two-dimensional array. When you specify 2, *grow*( ) adds a column to the array.

Property of

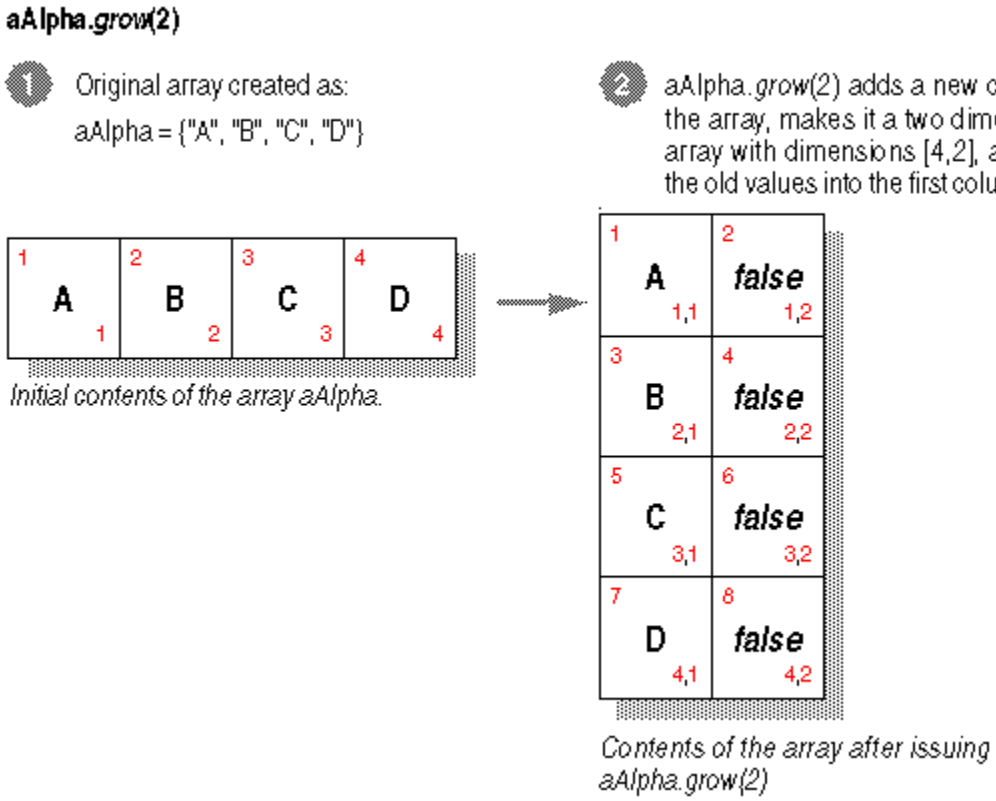
Array

Description

Use *grow*( ) to insert an element, row, or column into an array and change the size of the array to reflect the added elements. *grow*( ) can make a one-dimensional array two-dimensional. All added elements are initialized to *false* values.

One-dimensional arrays

When you specify 1 for <expN>, *grow*( ) adds a single element to the array. When you specify 2, *grow*( ) makes the array two-dimensional, and existing elements are moved into the first column. This is shown in the following figure:



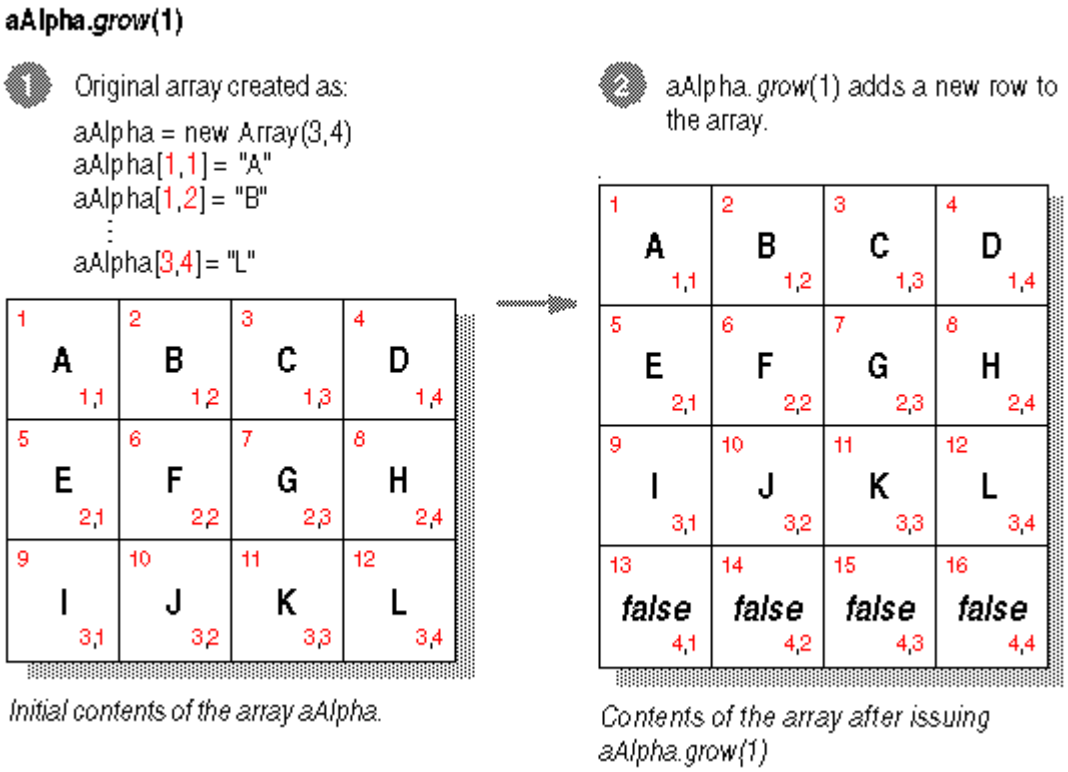
Use *add*( ) to add a new element to a one-dimensional array and assign its value in one step.

Note

You may also assign a new value to the array's *size* property to make a one-dimensional array any arbitrary size.

Two-dimensional arrays

When you specify 1 for <expN>, *grow*( ) adds a row to the array at the end of the array. This is shown in the following figure:



When you specify 2 for <expN>, *grow*( ) adds a column to the array and places *false* into each element in the column.

insert( )

Topic group Related topics Example

Inserts an element with the value *false* into a one-dimensional array, or inserts a row or column of elements with the value *false* into a two-dimensional array. Returns 1 if successful; generates an error if unsuccessful. The dimensions of the array do not change, so the element(s) at the end of the array will be lost.

Syntax

<oRef>.*insert*(<position expN> [, <row/column expN>])

<oRef>

A reference to a one- or two-dimensional array in which you want to insert data.

<position expN>

When <oRef> is a one-dimensional array, <position expN> specifies the number of the element in which you want to insert a *false* value.

When <oRef> is a two-dimensional array, <position expN> specifies the number of a row or column in which you want to insert *false* values. The second argument (discussed in the next paragraph) specifies whether <position expN> is a row or a column.

<row/column expN>

Either 1 or 2. If you omit this argument or specify 1, a row is inserted into a two-dimensional array. If you specify 2, a column is inserted. *dBASE Plus* generates an error if you use <row/column expN> with a one-dimensional array.

Property of

Array

Description

Use *insert*( ) to insert elements in an array. *insert*( ) does the following:

- Inserts an element in a one-dimensional array, or inserts a row or column in a two-dimensional array
- Moves all remaining elements toward the end of the array (down if a row is inserted, to the right if an element or column is inserted)
- Stores *false* values in the newly created position(s)

Because the dimensions of the array are not changed, the element(s) at the end of the array—the last element for a one-dimensional array or the last row or column for a two-dimensional array—are lost. If you don't want to lose the data, use *grow*( ) to increase the size of the array before using *insert*( ).

One-dimensional arrays

When you call *insert*( ) for a one-dimensional array, the logical value *false* is inserted into the position of the specified element. The remaining element(s) are moved one place toward the end of the array. The element that had been in the last position is lost.

For example, if you define a one-dimensional array with:

```
aAlpha = {"A", "B", "C"}
```

the resulting array has one row and can be illustrated as follows:

```
A B C
```

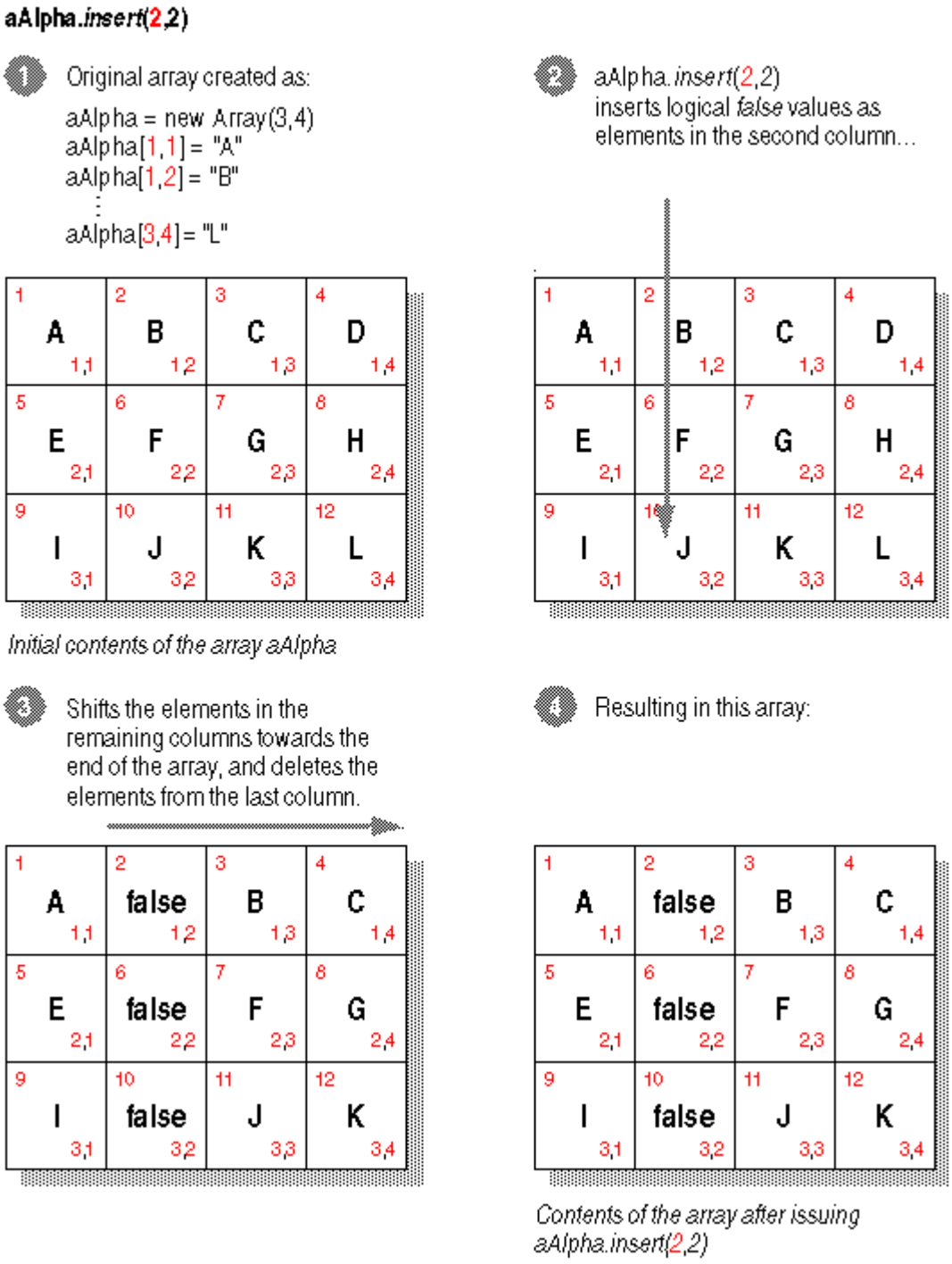
Issuing aAlpha.*insert*(2) inserts *false* into element number 2, moves the "B" that was in aAlpha[2] to aAlpha[3], and loses the "C" that was in aAlpha[3] so that the array now contains these values:

```
A false B
```

Two-dimensional arrays

When you call *insert*( ) for a two-dimensional array, a logical value *false* is inserted into the position of each element in the specified row or column. The elements in the remaining columns or rows are moved one place toward the end of the array. The elements that had been in the last row or column are lost.

For example, suppose you define a two-dimensional array and store letters to the array. The following illustration shows how the array is changed by aAlpha.*insert*(2,2).



isKey( )

Topic group Related topics Example

Returns a logical value that indicates if the specified character expression is the key of an element in an associative array.

Syntax

<oRef>.*isKey*(<expC>)

<oRef>

A reference to the associative array you want to search.

<expC>

The character string you want to find.

Property of

AssocArray

Description

Use *isKey*(<expC>) to determine if the specified character expression, <expC>, is the key of an element in an associative array. In associate arrays, an element's Key character expression is case-sensitive.

Attempting to access a non-existent key character expression in an associative array, will generate an error.

nextKey( )

Topic group Related topics Example

Returns the key name of the element following the specified key in an associative array.

Syntax

<oRef>.*nextKey*(<key expC>)

<oRef>

A reference to the associative array that contains the key.

<key expC>

An existing key name.

Property of

AssocArray

Description

Use *nextKey*( ) to loop through the elements in an associative array. Once you have determined the key name for the first element with *firstKey*, use *nextKey*( ) to get the key names for the rest of the elements.

nextKey( ) returns the key name for the key following <key expC>. Key names in associative arrays are case-sensitive. For the last key in the associative array and for a <key expC> that is not an existing key name, *nextKey*( ) returns the logical value *false*. Because *false* is a different data type than valid key names (which are character strings), it's difficult to look for *false* to terminate a loop. It's easier to get the number of elements in the array first with *count*( ); then loop through that many iterations.

Note

The order of elements in an associative array is undefined. They are not necessarily stored in the order in which you add them, or sorted by their key names. You can't assume that the sequence of keys will be consistent.

To determine if a given character string is a key name in an associative array, use *isKey*( ).

removeAll( )

Topic group Related topics Example

Deletes all elements from an associative array.

Syntax

<oRef>.*removeAll*( )

<oRef>

A reference to the associative array you want to empty.

Property of

AssocArray

Description

Use *removeAll*( ) to remove all the elements from an associative array.

To remove elements for particular key values, use *removeKey*( ).

removeKey( )

Topic group Related topics Example

Deletes an element from an associative array.

Syntax

<oRef>.*removeKey*(<key expC>)

<oRef>

A reference to the associative array that contains the key.

<key expC>

The key value of the element you want to delete.

Property of

AssocArray

Description

Use *removeKey*( ) to remove an element from an associative array. Key values in associative arrays are case-sensitive.

If you specify a key value that does not exist in the array, nothing happens; no error occurs and no elements are removed.

To remove all the elements from an associative array, use *removeAll*( ).

resize( )

Topic group Related topics Example

Sets the size of an array to the specified dimensions and returns a numeric value representing the number of elements in the modified array.

Syntax

<oRef>.*resize*(<rows expN> [,<cols expN> [, <retain values expN>]])

<oRef>

A reference to the array whose size you want to change.

<rows expN>

The number of rows the resized array should have. <rows expN> must always be a positive, nonzero value.

<cols expN>

The number of columns the resized array should have. <cols expN> must always be 0 or a positive value. If you omit this option, *resize*( ) changes the number of rows in the array and leaves the number of columns the same.

<retain values expN>

Determines what happens to the values of the array when rows are added or removed. If it is nonzero, values are retained. If you want to specify a value for <retain values expN>, you must also specify a value for <new cols expN>.

Property of

Array

Description

Use *resize*( ) to change the dimensions of an array, making it larger or smaller, or change the number of dimensions. To determine the number of dimensions, check the array's *dimensions* property. The *size* property of the array reflects the number of elements; for a one-dimensional array, that's all you need to know. For a two-dimensional array, you can't determine the number of rows or columns from the *size* property alone (unless the *size* is one—a one-by-one array). To determine the number of columns or rows in a two-dimensional array, use the ALEN( ) function.

For a one-dimensional array, you can change the number of elements by calling *resize*( ) and specifying the number of elements as <rows expN> parameter. You can also set the *size* property of the array directly, which is a bit less typing.

You can also change a one-dimensional array into a two-dimensional array by specifying both a <rows expN> and a nonzero <cols expN> parameter. This makes the array the designated size.

For a two-dimensional array, you can specify a new number of rows or both row and column dimensions for the array. If you omit <cols expN>, the <rows expN> parameter sets the number of rows only. With both a <rows expN> and a nonzero <cols expN>, the array is changed to the designated size.

You can change a two-dimensional array to a one-dimensional array by specifying <cols expN> as zero and <rows expN> as the number of elements.

To change the number of columns only for a two-dimensional array, you will need to specify both the <rows expN> and <cols expN> parameters, which means that you have to determine the number of rows in the array, if not known, and specify it unchanged as the <rows expN> parameter.

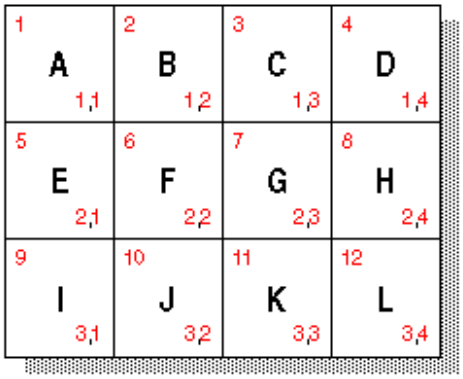
To add a single row or column to an array, use the *grow*( ) method.

If you add or remove columns from the array, you can use <retain values expN> to specify how you want existing elements to be placed in the new array. If <retain values *expN*> is zero or isn't specified, resize( ) rearranges the elements, filling in the new rows or columns or adjusting for deleted elements, and adding or removing elements at the end of the array, as needed. This is shown in the following two figures. You are most likely to want to do this if you don't need to refer to existing items in the array; that is, you plan to update the array with new values.

aAlpha.*resize*(4,5)

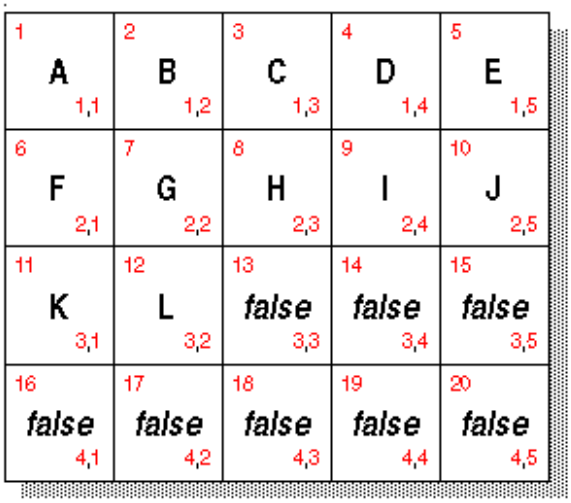
1 Original array created as:

```
aAlpha = new Array(3,4)
aAlpha[1,1] = "A"
aAlpha[1,2] = "B"
   ⋮
aAlpha[3,4] = "L"
```



Initial contents of the array aAlpha.

2 aAlpha.*resize*(4,5) adds a new row and column to the array and rearranges the values of the elements.

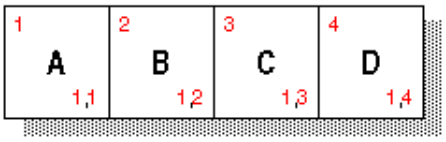


Contents of the array after issuing aAlpha.resize(4,5)

aAlpha.*resize*(,4,2)

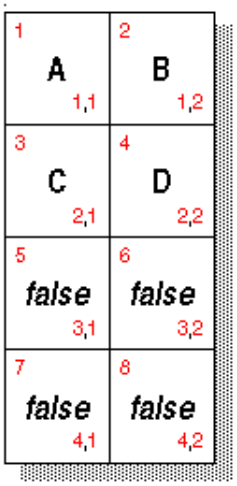
1 Original array created as:

```
aAlpha = {"A", "B", "C", "D"}
```



Initial contents of the array aAlpha.

2 aAlpha.*resize*(4,2) adds a new column to the array, makes it a two dimensional array with dimensions [4,2], and reassigns the values of the elements.



Contents of the array after issuing aAlpha.resize(4,2)

When you use *resize*( ) on a one-dimensional array, you might want the original row to become the first column of the new array. Similarly, when you use *resize*( ) on a two-dimensional array, you might want existing two-dimensional array elements to remain in their original positions. You are most likely to want to do this if you need to refer to existing items in the array by their

subscripts; that is, you plan to add new values to the array while continuing to work with existing values.

If <retain values expN> is a nonzero value, *resize*( ) ensures that elements retain their original values. The following two figures repeat the statements shown in the previous two figures, with the addition of a value of 1 for <retain values expN>.

aAlpha.*resize*(4,5,1)

1 Original array created as:

```
aAlpha = new Array(3,4)
aAlpha[1,1] = "A"
aAlpha[1,2] = "B"
   ⋮
aAlpha[3,4] = "L"
```

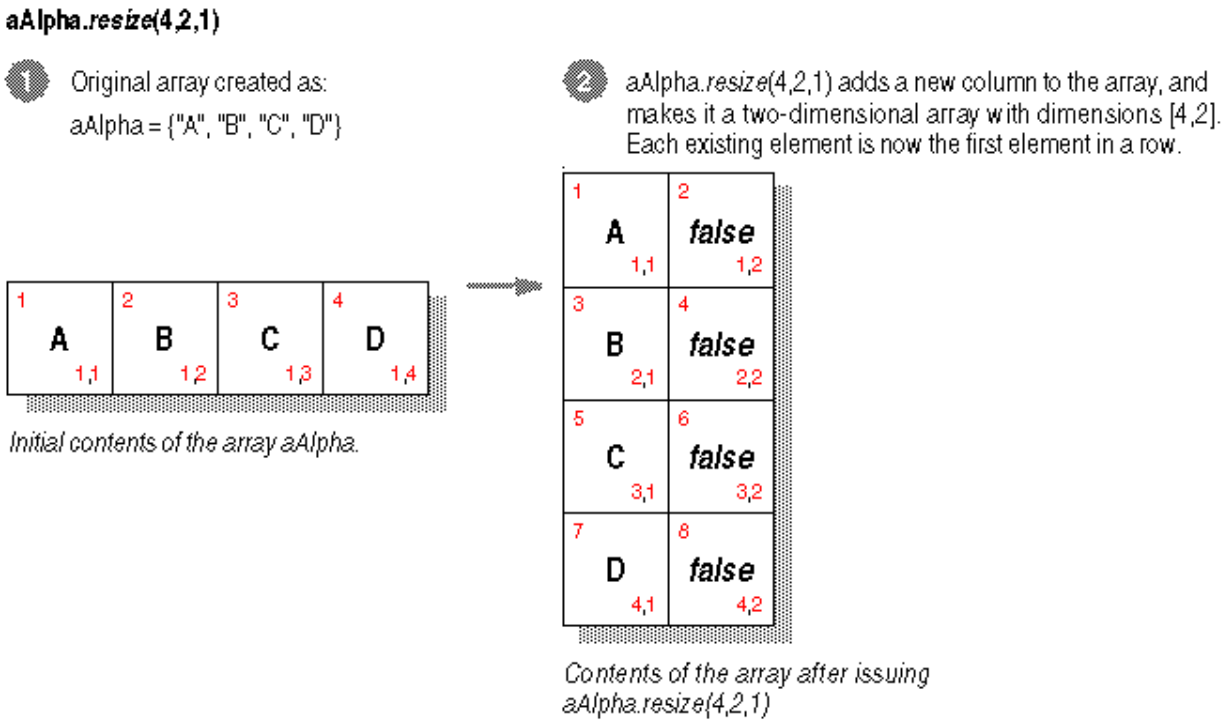
1 A 1,1	2 B 1,2	3 C 1,3	4 D 1,4
5 E 2,1	6 F 2,2	7 G 2,3	8 H 2,4
9 I 3,1	10 J 3,2	11 K 3,3	12 L 3,4

Initial contents of the array aAlpha.

2 aAlpha.*resize*(4,5,1) adds a new row and column to the array and maintains the values of the elements.

1 A 1,1	2 B 1,2	3 C 1,3	4 D 1,4	5 *false* 1,5
6 E 2,1	7 F 2,2	8 G 2,3	9 H 2,4	10 *false* 2,5
11 I 3,1	12 J 3,2	13 K 3,3	14 L 3,4	15 *false* 3,5
16 *false* 4,1	17 *false* 4,2	18 *false* 4,3	19 *false* 4,4	20 *false* 4,5

Contents of the array after issuing aAlpha.resize(4,5,1)



scan( )

Topic group Related topics Example

Searches an array for an expression. Returns the number of the first element that matches the expression if the search is successful, or 0 if the search is unsuccessful.

Syntax

<oRef>.*scan*(<exp> [, <starting element expN> [, <elements expN>]])

<oRef>

A reference to the array you want to search.

<exp>

The expression you want to search for in <oRef>.

<starting element expN>

The element number in <oRef> at which you want to start searching. Without <starting element expN>, *scan*( ) starts searching at the first element.

<elements expN>

The number of elements in <oRef> that *scan*( ) searches. Without <elements expN>, *scan*( ) searches <oRef> from <starting element expN> to the end of the array. If you want to specify a value for <elements expN>, you must also specify a value for <starting element expN>.

Property of

Array

Description

Use *scan*( ) to search an array for the value of <exp>. For example, if an array contains customer names, you can use *scan*( ) to find the location in which a particular name appears.

scan( ) returns the element number of the first element in the array that matches <exp>. If you want to determine the subscripts of this element, use *subscript*( ).

When <exp> is a string, *scan*( ) is case-sensitive; you may want to use UPPER( ), LOWER( ), or PROPER( ) to match the case of <exp> with the case of the data stored in the array. *scan*( ) also follows the rules established by SET EXACT to determine if <exp> and the array element are equal. For more information, see SET EXACT.

size

Topic group Related topics Example

The number of elements in an Array object.

Property of

Array

Description

size indicates the number of elements in an Array object.

For a one-dimensional array, you can assign a value to *size* to change its size.

For an array with more than one dimension, *size* is read-only.

You can use the ALEN( ) function to determine the size of each dimension for a two-dimensional array. There is no built-in way to determine dimension sizes for arrays with more than two dimensions.

sort( )

Topic group Related topics Example

Sorts the elements in a one-dimensional array or the rows in a two-dimensional array. Returns 1 if successful; generates an error if unsuccessful.

Syntax

<oRef>.*sort*([<starting element expN> [,<elements to sort expN> [, <sort order expN>]]])

<oRef>

A reference to the array you want to sort.

<starting element expN>

In a one-dimensional array, the number of the element in <oRef> at which you want to start sorting. In a two-dimensional array, the number (subscript) of the column on which you want to sort. Without <starting element expN>, *sort*( ) starts sorting at the first element or column in the array.

<elements to sort expN>

In a one-dimensional array, the number of elements you want to sort. In a two-dimensional array, the number of rows to sort. Without <elements to sort expN>, *sort*( ) sorts the rows starting at the row containing element <starting element expN> to the last row. If you want to specify a value for <elements to sort expN>, you must also specify a value for <starting element expN>.

<sort order expN>

The sort order:

0 specifies ascending order (the default)

1 specifies descending order

If you want to specify a value for <sort order expN>, you must also specify values for <elements to sort expN> and <starting element expN>.

Property of

Array

Description

sort( ) requires that all the elements on which you're sorting be of the same data type. The elements to sort in a one-dimensional array must be of the same data type, and the elements of the column by which rows are to be sorted in a two-dimensional array must be of the same data type.

sort( ) arranges elements in alphabetical, numerical, chronological, or logical order, depending on the data type of <starting element expN>. (For strings, the current language driver determines the sort order.)

One-dimensional arrays

Suppose you create an array with the following statement:

```
aNums = {5, 7, 3, 9, 4, 1, 2, 8}
```

That creates an array with the elements in this order:

```
5 7 3 9 4 1 2 8
```

If you call aNums.*sort*(1, 5), *dBASE Plus* sorts the first five elements so that the array elements are in this order:

```
3 4 5 7 9 1 2 8
```

If you then call aNums.*sort*(5, 2), *dBASE Plus* sorts two elements starting at the fifth element so that the array elements are now in this order:

```
3 4 5 7 1 9 2 8
```

Two-dimensional arrays

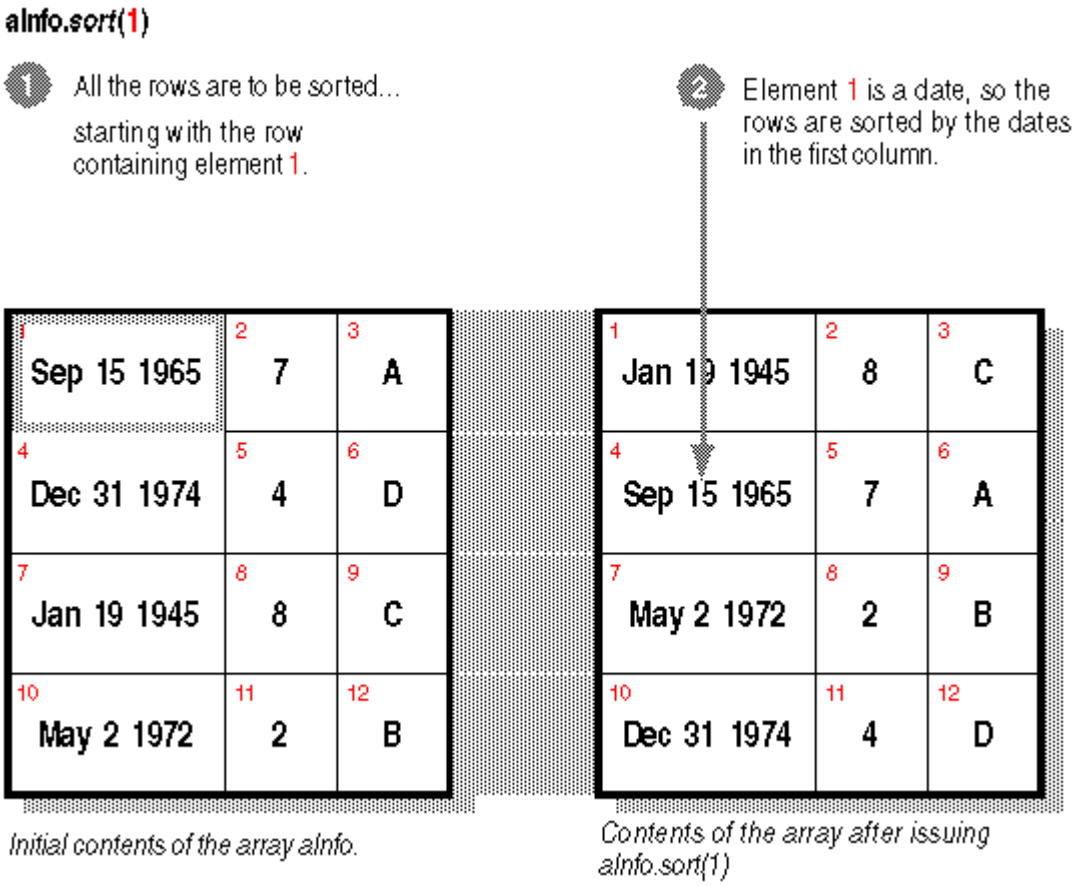
Using *sort*( ) with a two-dimensional array is similar to using the SORT command with a table. Array rows correspond to records, and array columns correspond to fields.

When you sort a two-dimensional array, whole rows are sorted, not just the elements in the column where <starting element expN>) is located.

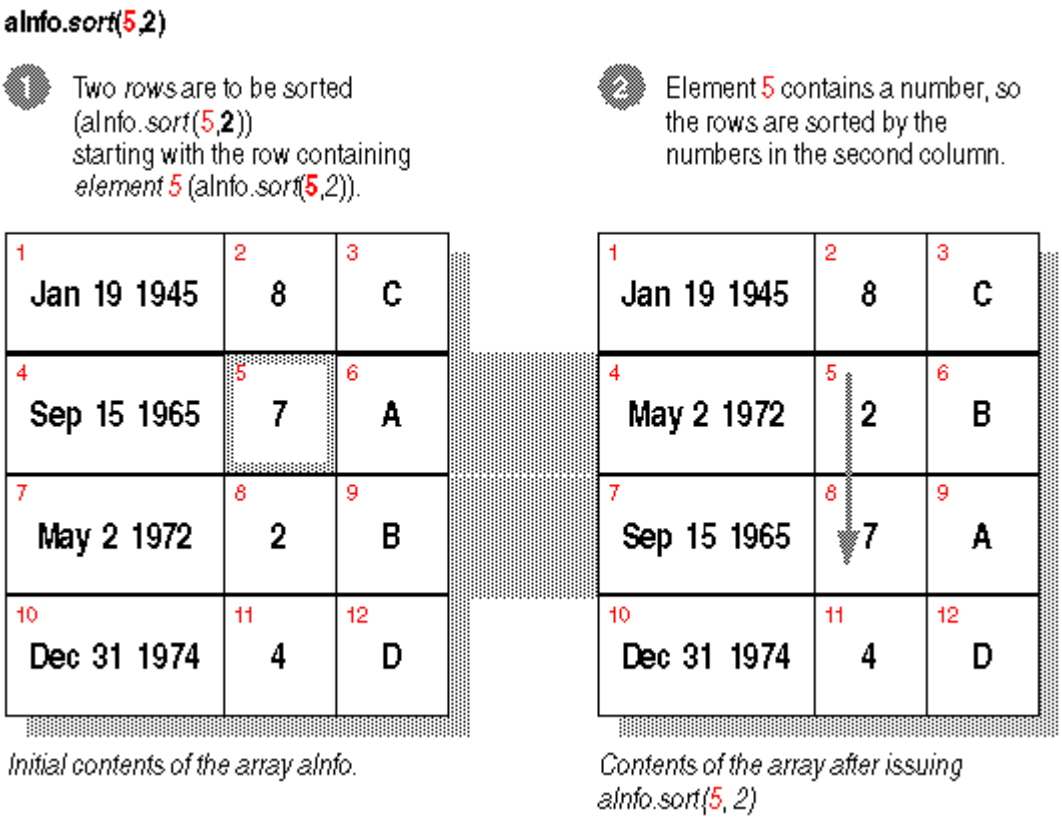
For example, suppose you create the array aInfo and fill it with the following data:

Sep 15 1965	7	A
Dec 31 1965	4	D
Jan 19 1945	8	C
May 2 1972	2	B

If you call aInfo.*sort*(1), *dBASE Plus* sorts all rows in the array beginning with element number 1. The rows are sorted by the dates in the first column because element 1 is a date. The following figure shows the results.



If you then call aInfo.*sort*(5, 2), *dBASE Plus* sorts two rows in the array starting with element number 5, whose value is 7. *sort*( ) sorts the second and the third rows based on the numbers in the second column. The following figure shows the results.



subscript( )

Topic group Related topics Example

Returns the row number or the column number of a specified element in an array.

Syntax

<oRef>.*subscript*(<element expN>, <row/column expN>)

<oRef>

A reference to the array.

<element expN>

The element number.

<row/column expN>

A number, either 1 or 2, that determines whether you want to return the row or column subscript of an array. If <row/column expN> is 1, subscript( ) returns the number of the row subscript. If <row/column expN> is 2, subscript( ) returns the number of the column subscript.

If <oRef> is a one-dimensional array, *dBASE Plus* returns an error if <row/column expN> is a value other than 1.

Property of

Array

Description

Use *subscript*( ) when you know the number of an element in a two-dimensional array and want to reference the element by using its subscripts.

If you need to determine both the row and column number of an element in a two-dimensional array, call *subscript*( ) twice, once with a value of 1 for <row/column expN> and once with a value of 2 for <row/column expN>. For example, if the element number is in the variable nElement, execute the following statements to get its subscripts:

```
nRow = aExample.subscript( nElement, 1 )
nCol = aExample.subscript( nElement, 2 )
```

In one-dimensional arrays, the number of an element is the same as its subscript, so there is no need to use *subscript*( ). For example, if aOne is a one-dimensional array, aOne.*subscript*(3) returns 3, aOne.*subscript*(5) returns 5, and so on.

subscript( ) is the inverse of *element*( ), which returns an element number when you specify the element subscripts.

Date and Time

Date and time

Topic group

dBASE Plus supports two types of dates:

- A primitive date that is compatible with earlier versions of dBASE
- A JavaScript-compatible Date object.

A Date object represents a moment in time. It is stored as the number of milliseconds since January 1, 1970 00:00:00 GMT (Greenwich Mean Time). Although GMT and UTC (a compromise between the English and French acronyms for Universal Coordinated Time) are derived differently, they are considered to represent the same time.

Modern operating systems have their own current time zone setting, which is used when handling Date objects. For example, two computers with different time zone settings—whether or not they are physically in different time zones—will display the same time differently.

Primitive dates represent the date only, not the time. (They are considered to be the first millisecond—midnight—of that date.) Literal dates are delimited by curly braces and are evaluated according to the rules used by the CTOD( ) function. An invalid literal date is always converted to the next valid one; for example, if the current date format is month/day/year, {02/29/1997} is considered March 1, 1997. An empty date is valid and is represented by empty braces: { }.

dBASE Plus will convert one type of date to the other on-the-fly as needed. For example, you may use a Date class method on a primitive date variable or a literal date:

```
? date( ).toGMTString( )
? {8/21/97}.toString( )
```

This creates a temporary Date object from which the method or property is called. Because the object is a temporary copy, calling the set methods or assigning to the properties is allowed, but has no apparent effect. You may also use a date function on a Date object, in which case the time portion of the Date object will be truncated.

Note

While the JavaScript-compatible methods are zero-based, dBL functions are one-based. For example, the *getMonth*( ) method returns 0 for January, while MONTH( ) returns 1.

dBASE Plus also features a Timer object that can cause actions to occur at timed intervals.

class Date

Topic group

An object that represents a moment in time.

Syntax

[<oRef> =] new Date( )

or

[<oRef> =] new Date(<date expC>)

or

[<oRef> =] new Date(<msec expN>)

or

[<oRef> =] new Date(<year expN>, <month expN>, <day expN>
[, <hours expN> , <minutes expN> , <seconds expN>])

or

[<oRef> =] new Date(<year expN>, <month expN>, <day expN>
[, <hours expN> , <minutes expN> , <seconds expN>, <timez expC>])

<oRef>

A variable or property in which you want to store a reference to the newly created Date object.

<date expC>

A string representing a date and time.

<msec expN>

The number of milliseconds since January 1, 1970 00:00:00 GMT. Negative values can be used for dates before 1970.

<year expN>

The year.

<month expN>

A number representing the month, between 0 and 11: zero for January, one for February, and so on, up to 11 for December.

<day expN>

The day of the month, from 1 to 31.

<hours expN>

The hours portion of the time, from 1 to 24.

<minutes expN>

The minutes portion of the time, from 1 to 60.

<seconds expN>

The seconds portion of the time, from 1 to 60.

<timez expC>

A Time Zone (GMT, EST, CST, MST or PST)

Properties

The following tables list the properties and methods of the Date class. (No events are associated with this class.).

Property	Default	Description
baseClassName	DATE	Identifies the object as an instance of the Date class
className	(DATE)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
date		The day of the month
day		The day of the week, from 0 to 6: 0=Sunday, 6=Saturday
hour		The hour component of time (**9**:18:34)
minute		The minute component of time (9:**18**:34)
month		The month of the year, from 0 to 11: 0=January, 11=December
second		The seconds component of time (9:18:**34**)
year		The year of the date

Method	Parameters	Description
getDate( )		Returns day of month
getDay( )		Returns day of week
getHours( )		Returns hours portion of time
getMinutes( )		Returns minutes portion of time
getMonth( )		Returns month of year
getSeconds( )		Returns seconds portion of time
getTime( )		Returns date/time equivalent
getTimezoneOffset( )		Returns time zone offset for current locale
getYear( )		Returns year of date
parse( )	<date expC>	Calculates time equivalent for date string
setDate( )	<expN>	Sets day of month
setHours( )	<expN>	Sets hours portion of time
setMinutes( )	<expN>	Sets minutes portion of time
setMonth( )	<expN>	Sets month of year
setSeconds( )	<expN>	Sets seconds portion of time
setTime( )	<expN>	Sets date/time
setYear( )	<expN>	Sets year of date
toGMTString( )		Converts date to string, using Internet (GMT) conventions
toLocaleString( )		Converts date to string, using locale conventions
toString( )		Converts date to string, using standard JavaScript conventions
UTC( )	<year expN>, <month expN>, <day expN>, [<hours expN>, <minutes expN>, <seconds expN>]	Calculates time equivalent of date parameters

Description

A Date object represents both a date and time.

There are four ways to create a new Date object:

When called with no parameters, the Date object contains the current system date and time.

You can pass a string containing a date and optionally a time. Once a time parameter has been specified, the time zone parameter may also be included. Lacking a time zone parameter, *dBASE Plus* defaults to the current locale.

You can pass a number representing the number of milliseconds since January 1, 1970, 00:00:00 GMT. Use a negative number for dates before 1970.

You can pass numeric parameters for each component of the date, and optionally each component of the time.

If you specify a date but don't specify hours, minutes, or seconds, they are set to zero. When passing a string, the *<date expC>* can be in a variety of formats, with or without the time, as shown in the following examples:

```
d1 = new Date( "Jan 5 1996" ) // month, day, year
d2 = new Date( "18 Dec 1994 15:34" ) // day, month, year, and time
d3 = new Date( "1987 Nov 4 9:18:34" ) // year, month, day, and time with seconds
d4 = new Date( "1987 Nov 4 9:18:34 PST" ) // year, month, day, time with seconds,
and time zone
```

You may spell out the month or abbreviate it, down to the first three letters; for example, "April", "Apri", or "Apr". For consistency and because of the three-letter month of May, you should either always spell it out completely or use the first three letters.

Date objects have an inherent value. The format of the date is platform-dependent; in *dBASE Plus*, the format is the same as using the *toLocaleString*( ) method. Use the *toGMTString*( ), *toLocaleString*( ), and *toString*( ) methods to format the Date objects, or create your own. Date objects will automatically type-convert into strings, using the inherent format.

In *dBASE Plus*, every Date object has a separate property for each date and time component. You may read or write to these properties directly (except for the *day* property, which is read-only), or use the equivalent method. For example, assigning a value to the *minute* property has the same effect as calling the *setMinutes*( ) method with the value as the parameter.

Note

While using values outside a date component's specified range does not produce an error message, they may produce unintended results. In the following example, an inadvertant minus sign before the hours component actually rolls the clock back:

```
d=new date(01,05,13,23,20,30)
?dtodt(d)
06/13/2001 11:20:30 PM

d=new date(01,05,13,-23,20,30)
?dtodt(d)
06/12/2001 01:20:30 AM
```

Change the month to 12 and watch the result jump to:

```
01/12/2002 01:20:30 AM
```

To avoid such scenarios, it is recommended that date component values fall within their stated range.

You should also aquaint yourself with the affect "rollover" will have on your date components. With the exception of the month component, "rollover" occurs whenever you use the highest number in a range. For example, using 60 for the seconds value will cause the minutes value to increase by 1, 60 minutes rollsovers to the next hour, and so on.

class Timer

Topic group Example

An object that initiates a recurring action at preset intervals.

Syntax

[<oRef> =] new Timer( )

<oRef>

A variable or property in which you want to store a reference to the newly created Timer object.

Properties

The following tables list the properties and events of the Timer class. (No methods are associated with this class.) For details on each property, click on the property below.

Property	Default	Description
baseClassName	TIMER	Identifies the object as an instance of the Timer class
className	(TIMER)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
enabled	false	Whether the Timer is active
interval	10	The interval between actions, in seconds

Event	Parameters	Description
onTimer		Action to take when *interval* expires

Description

To use a Timer object:

1. Assign an event handler to the *onTimer* event.
    - Set the *interval* property to the desired number of seconds.
    - Set the *enabled* property to *true* when you want to activate the timer.

The Timer object will start counting down time whenever *dBASE Plus* is idle. When the number of seconds assigned to *interval* has passed, the Timer object's *onTimer* event fires. After the event fires, the Timer object's internal timer is reset back to the *interval*, and the countdown repeats.

To disable the timer, set the *enabled* property to *false*.

A Timer object counts idle time; that is when *dBASE Plus* is not doing anything. This includes waiting for input in the Command window or Navigator. If a process, such as an event handler or program, is running, the counter in all active Timer objects is suspended. When the process is complete and *dBASE Plus* is idle again, the count resumes.

CDOW( )

Topic group Related topics Example

Returns the name of the day of the week of a specified date.

Syntax

CDOW(<expD>)

<expD>

The date whose corresponding weekday name to return.

Description

CDOW( ) returns a character string containing the name of the day of the week on which a date falls. To return the day of the week as a number from 1 to 7, use DOW( ).

If you pass an blank or invalid date to CDOW( ), it returns "Unknown".

CMONTH( )

Topic group Related topics Example

Returns the name of the month of a specified date.

Syntax

CMONTH(<expD>)

<expD>

The date whose corresponding month name to return.

Description

CMONTH( ) returns a character string containing the name of the month in which a date falls. To return the month as a number from 1 to 12, use MONTH( ).

If you pass an blank or invalid date to CMONTH( ), it returns "Unknown".

CTOD( )

Topic group Related topics Example

Interprets a specified character expression as a literal date.

Syntax

CTOD(<expC>)

<expC>

The character expression, in the current date format, to return as a date.

Description

Use CTOD( ) to convert a character expression containing a literal date to a date value. Once you convert the string to a date, you can manipulate it with date functions and date arithmetic.

A literal date must be in format:

```
<number><separator><number><separator><number>[BC]
```

where <separator> should be a slash (/), hyphen (-), or period (.). The two <separator> characters should match. You may specify a BC date by including the letters "BC" (not case-sensitve) at the end of the literal date.

To specify a literal date in code, use curly braces ({ }) as literal date delimiters; there is no need to use CTOD( ). For example, there two are equivalent:

```
{04/05/06}
ctod( "04/05/06" )
```

The interpretation of the literal date—that is, which numbers are the day, month, and year, and how two-digit years are handled—is controlled by the current settings for SET DATE and SET EPOCH. For example, if SET DATE is MDY and SET EPOCH is 1930, the literal date above is April 5, 2006.

SET DATE also controls the display of dates, while SET EPOCH does not. SET CENTURY controls the display of dates, but has no effect on how dates are interpreted. Two-digit years are always treated as years in the current epoch.

If you pass an invalid date to CTOD( ), it attempts to convert the date to a valid one. For example, it interprets June 31 (June only has 30 days) as July 1. If you pass an empty or non-literal-date string to CTOD( ), it returns an blank date, which is a valid date value.

CTODT( )

Topic group Related topics Example

"Character to DateTime" converts a literal DateTime string to a DateTime (DT) value.

Syntax

CTODT(<expC>)

<expC>

The character expression, in the current DateTime format, to return as a DateTime value.

Description

Use CTODT( ) to convert a DateTime string to a DateTime value. DateTime values are their own type (DT).

SET DATE determines the order of the day, month, and year.

SET CENTURY determines whether the year is expressed as two or four digits.

SET MARK assigns the separator character.

SET HOURS determines whether times are displayed in military format, or with an AM/PM indicator.

CTOT( )

Topic group Related topics

"Character to TIME"( ) converts a literal Time string to a Time value.

Syntax

CTOT(<expC>)

<expC>

The character expression, in the current Time format, to return as a Time value.

Description

Use CTOT( ) to convert a Time string to a Time value. Time strings returned by the Time( ) function result in an HH:MM:SS, military time format. When these strings are converted to values, through the CTOT( ) function, the result can be displayed with an attached AM/PM indicator when SET HOURS is set to 12.

When determining the duration between two events, subtracting the earlier from the later produces the lapsed time displayed in seconds.

DATE( )

Topic group Related topics Example

Returns the system date.

Syntax

DATE( )

Description

DATE( ) returns your computer system's current date.

To change the system date, use SET DATE TO.

DATETIME( )

Topic group Related topics Example

Returns a value representing the current date and time.

Syntax

DATETIME( )

Description

Use DATETIME( ) to determine the lapsed time between two or more events. The actual value of DATETIME( ) appears internally in scientific notation as fractions of days, and provides little in the way of visually relevant information. Subtracting the current DATETIME( ) from another a short while later could produce something resembling -.92245370370436E-4.

To use DATETIME( ) values in a more practical format, convert the value to a character string and extract the date and/or time elements. DATETIME( ) values can be converted to character strings using the DTTOC( ) function (DateTime to Character), and back to values using the CTODT( ) function (Character to DateTime).

Once the date and time character strings have been extracted, you can convert the resulting strings to values using the CTOD( ) or CTOT( ) functions, and back again using DTOC ( ) or TTOC( ) respectively.

If you are utilizing a timestamp field you could store the current date and time to a field defined as a timestamp type:

```
queryName.rowset.fields["timestampfield"].value = DATETIME( )
```

DAY( )

Topic group Related topics Example

Returns the numeric value of the day of the month for a specified date.

Syntax

DAY(<expD>)

<expD>

The date, or DateTime, whose corresponding day-of-the-month number you want to return.

Description

DAY( ) returns a day of the month number—a value from 1 to 31—for a specified date or DateTime .

DAY( ) returns zero for a blank date.

DMY( )

Topic group Related topics

Returns a specified date as a character string in DD MONTH YY or DD MONTH YYYY format.

Syntax

DMY(<expD>)

<expD>

The date to format.

Description

DMY( ) returns a date in DD MONTH YY or DD MONTH YYYY format, where DD is the day number, MONTH is the full month name, and YY is the year number. If SET CENTURY is OFF (the default), DMY( ) returns the year as 2 digits. If SET CENTURY is ON, DMY( ) returns the year as 4 digits. If the day is only one digit, it is preceded by a space.

If you pass an blank date to DMY( ), it returns "0 Unknown 00" or "0 Unknown 0000".

DOW( )

Topic group Related topics Example

Returns the day of the week corresponding to a specified date as a number from 1 to 7.

Syntax

DOW(<expD>)

<expD>

The date whose corresponding weekday number you want to return.

Description

DOW( ) returns the number of the day of the week on which a date falls:

Day	Number
Sunday	1
Monday	2
Tuesday	3
Wednesday	4
Thursday	5
Friday	6
Saturday	7

To return the name of the day of the week instead of the number, use CDOW( ). DOW( ) returns zero for a blank date.

DTOC( )

Topic group Related topics Example

Converts a date into a literal date string.

Syntax

DTOC(<expD>)

<expD>

The date to return as a string.

Description

There are many different ways to represent a date as a string. Use DTOC( ) to convert a date into a literal date string, one that is suitable for conversion back into a date by CTOD( ).

The order of the day, month, and year is controlled by the current SET DATE setting. Whether the year is expressed as two or four digits is controlled by SET CENTURY. The separator character is controlled by SET MARK.

Note

To convert a date expression to a character string suitable for indexing or sorting, always use DTOS( ), which converts the date into a consistent and sortable format.

If you pass a blank date to DTOC( ), it returns a string with spaces instead of digits. For example, if the SET DATE format is AMERICAN and SET CENTURY is OFF, DTOC({ }) returns " / / ".

When concatenating a date to a string, *dBASE Plus* automatically converts the date using DTOC( ) for you.

DTODT( )

Topic group Related topics

"Date to DateTime" converts a date to a DateTime value (DT).

Syntax

DTODT(<expD>)

<expD>

The date to return as a DateTime value.

Description

Use DTODT( ) to convert a date into DateTime value. DateTime values are their own type (DT). DTODT( ) only affects the date component of the DateTime value. The time component is displayed as 12:00:00 AM when SET HOURS is set to 12, and 00:00:00 when SET HOURS is set to 24. Where the current date is 12/25/2001;

```
d1=date()
d2=DTODT(D1)  //Yields 12/25/2001 12:00:00 AM (SET HOURS=12)
          OR 12/25/2001 00:00:00 (SET HOURS=24)
```

SET DATE determines the order of the day, month, and year.

SET CENTURY determines whether the year is expressed as two or four digits.

SET MARK assigns the separator character.

DTOS( )

Topic group Related topics Example

Returns a specified date as a character string in YYYYMMDD format.

Syntax

DTOS(<expD>)

<expD>

The date expression to return as a character string in YYYYMMDD format.

Description

Use DTOS( ) to convert a date expression to a character string suitable for indexing or sorting. For example, you can use DTOS( ) when indexing on a date field in combination with another field of a different type. DTOS( ) always returns a character string in YYYYMMDD format, even if SET CENTURY is OFF.

If you pass a blank date to DTOS( ), it returns a string with eight spaces, which matches the length of the normal result.

DTTOC( )

Topic group Related topics

"DateTime to Character" converts a DateTime value to a literal DateTime string.

Syntax

DTTOC(<dtVar>)

<dtVar>

A DateTime variable or value

Description

Use DTTOC( ) to convert a DateTime value into a literal DateTime string.

The order of the day, month, and year is controlled by the current SET DATE setting. Whether the year is expressed as two or four digits is controlled by SET CENTURY. The separator character is controlled by SET MARK.

Once the DateTime value has been converted to a character string, it's integral parts, DATE and TIME, can be extracted using the LEFT( ) or RIGHT( ) functions. When SET CENTURY is OFF, the date and time strings can be extracted using

```
left("value",8) and right("value",11) respectively.
```

Note

To recombine extracted date and time values into a DateTime format, see CTODT( ) (Character to DateTime)

DTTOD( )

Topic group Related topics

"DateTime to Date" converts the date component of a DateTime value to a literal Date .

Syntax

DTTOD(<dtVar>)

<dtVar>

A DateTime variable or value

Description

Use DTTOD( ) to convert the date component of a DateTime value into a literal Date. DTTOC( ) has no affect on the DateTime's time component. Where the current value of DATETIME( ) = 02/13/01 03:39:14 PM:

```
d1=DATETIME()
d2=DTTOD(d1)
?d2      //Yields 02/13/01
```

SET DATE determines the order of the day, month, and year.

SET CENTURY determines whether the year is expressed as two or four digits.

SET MARK assigns the separator character.

DTTOT( )

Topic group Related topics

"DateTime to Time" converts the time component of a DateTime value to a Time value .

Syntax

DTTOT(<dtVar>)

<dtVar>

A DateTime variable or value

Description

Use DTTOT( ) to convert the time component of a DateTime value to a Time value. DTTOT( ) has no affect on the DateTime's date component. Where the current value of DATETIME( ) = 02/13/01 03:39:14 PM:

```
t1=DATETIME()
t2=DTTOT(t1)
?t2      //Yields 03:39:14 PM
```

SET HOURS determines whether times are displayed in military format, or with an AM/PM indicator.

ELAPSED( )

Topic group Related topics Example

Returns the number of seconds elapsed between two specified times.

Syntax

ELAPSED(<stop time expC>, <start time expC> [, <exp>])

<stop time expC>

The time expression, in the format HH:MM:SS, at which to stop timing seconds elapsed. The <stop time expC> argument should be a later time than <start time expC>; if it is not, *dBASE Plus* returns a negative value.

<start time expC>

The time expression, in the format HH:MM:SS, at which to start timing seconds elapsed. The <start time expC> argument should be an earlier time than <stop time expC>; if it is not, *dBASE Plus* returns a negative value.

<exp>

Any expression, which causes ELAPSED( ) to calculate hundredths of a second. The format of both <start time expC> and <stop time expC> can be HH:MM:SS.hh.

Description

Use ELAPSED( ) with TIME( ) to time a process. Call TIME( ) at the start of the process and store the resulting time string to a variable. Then call TIME( ) again at the end of the process. Call ELAPSED( ) with the start and stop times to calculate the number of seconds between.

ELAPSED( ) subtracts the value of <start time expC> from <stop time expC>. If <start time expC> is the later time, ELAPSED( ) returns a negative value. Both <stop time expC> and <start time expC> must be in HH:MM:SS or HH:MM:SS.hh format, where HH is the hour, MM the minutes, SS the seconds, and hh is hundredths of a second.

Without <exp>, any hundredths of a second are truncated and ignored; ELAPSED( ) does not round hundredths of a second when <exp> is omitted.

enabled

Topic group Related topics Example

Specifies whether a Timer object is active and counting down time.

Property of

Timer

Description

Set the *enabled* property to *true* to activate the Timer object. When the number of seconds of idle time specified in the *interval* property has passed, the timer's *onTimer* event fires.

When the *enabled* property is set to *false*, the Timer stops counting time and the internal counter is reset. For example, suppose that

1. The *interval* is 10, and *enabled* is set to *true*.
    - Then 9 seconds of idle time go by, and
    - *enabled* is set to *false*.

If the *enabled* property is set to *true* again, the *onTimer* event will fire after another 10 seconds has gone by, even though there was only 1 second left before the timer was disabled.

If a Timer is intended to go off only once, instead of repeatedly, set the *enabled* property to *false* in the *onTimer* event handler.

getDate( )

Topic group Related topics Example

Returns the numeric value of the day of the month.

Syntax

<oRef>.*getDate*( )

<oRef>

The Date object whose corresponding day-of-the-month number you want to return.

Property of

Date

Description

The *getDate*( ) method returns a date's day of the month number—a value from 1 to 31. If the Date object contains a blank date, the *getDate*( ) method returns 0.

getDay( )

Topic group Related topics Example

Returns the day of the week corresponding to a specified date as a number from 0 to 6.

Syntax

<oRef>.*getDay*( )

<oRef>

The Date object whose corresponding weekday number you want to return.

Property of

Date

Description

The *getDay*( ) method returns the number of the day of the week on which a date falls. The number is zero-based:

Day	Number
Sunday	0
Monday	1
Tuesday	2
Wednesday	3
Thursday	4
Friday	5
Saturday	6

Note

The equivalent date function, DOW( ), is one-based, not zero-based.

The day of the week is the only date/time component you cannot set directly; there is no corresponding set- method. It is always based on the date itself.

getHours( )

Topic group Related topics Example

Returns the hours portion of a date object.

Syntax

<oRef>.*getHours*( )

<oRef>

The date object whose hours you want to return.

Property of

Date

Description

The *getHours*( ) method returns the hours portion of the time (using a 24-hour clock) in a Date object: an integer from 0 to 23.

getMinutes( )

Topic group Related topics

Returns the minutes portion of a date object.

Syntax

<oRef>.getMinutes( )

<oRef>

The date object whose minutes you want to return.

Property of

Date

Description

The *getMinutes*( ) method returns the minutes portion of "time" from a Date object: an integer from 0 to 59.

getMonth( )

Topic group Related topics

Returns the number of the month for a specified date.

Syntax

<oRef>.*getMonth*( )

<oRef>

The Date object whose corresponding month number you want to return.

Property of

Date

Description

The *getMonth*( ) method returns a date's month number. The number is zero-based:

Month	Number
January	0
February	1
March	2
April	3
May	4
June	5
July	6
August	7
September	8
October	9

November	10
December	11

Note

The equivalent date function, MONTH( ), is one-based, not zero-based.

getSeconds( )

Topic group Related topics

Returns the seconds portion of a date object.

Syntax

<oRef>.*getSeconds*( )

<oRef>

The date object whose seconds you want to return.

Property of

Date

Description

The *getSeconds*( ) method returns the seconds portion of "time" from a Date object: an integer from 0 to 59.

getTime( )

Topic group Related topics Example

Returns time equivalent of date/time, in milliseconds.

Syntax

<oRef>.*getTime*( )

<oRef>

The Date object whose time equivalent you want to return.

Property of

Date

Description

The *getTime*( ) method returns the number of milliseconds since January 1, 1970 00:00:00 GMT for the date/time stored in the Date object. All date/times are represented internally by this millisecond number.

getTimezoneOffset( )

Topic group Related topics

Returns the time zone offset for a date object in the current locale, in minutes.

Syntax

<oRef>.*getTimezoneOffset*( )

<oRef>

A date object created in the locale in question.

Property of

Date

Description

All time zones have an offset from GMT (Greenwich Mean Time), from twelve hours behind to twelve hours ahead. *The getTimezoneOffset*( ) method returns this offset, in minutes, for the locale in which the Date object was created, taking Daylight Savings Time into account.

For example, the United States and Canadian Pacific time zone is eight hours behind GMT. A date in January, when Daylight Savings Time is not in effect, created in the Pacific time zone would have a time zone offset of –480. A date in July, when Daylight Savings Time is in effect, would have a time zone offset of –420, or seven hours, since Daylight Savings Time moves clocks one hour forward, or closer, to GMT.

In Windows, the locale is determined by the Time Zone setting in each system's Date/Time properties. This can be set in the Control Panel, or by double-clicking the clock in the Taskbar.

All Date objects default to the time zone setting of the current locale.

getYear( )

Topic group Related topics

Returns the year of a specified date.

Syntax

<oRef>.*getYear*( )

<oRef><expD>

The Date object whose corresponding year number you want to return.

Property of

Date

Description

The *getYear*( ) method returns a date's "year" number. A 4-digit year is always returned. The SET CENTURY setting has no effect on the *getYear*( ) method.

interval

Topic group Related topics Example

The amount of idle time, in seconds, between the firings of the timer.

Property of

Timer

Description

Set the *enabled* property to *true* to activate the Timer object. When the number of seconds of idle time specified in the *interval* property has passed, the timer's *onTimer* event fires.

When the *enabled* property is set to *false*, the Timer stops counting time and the internal counter is reset. For example, suppose that

1. The *interval* property is set to 10, and the *enabled* property is set to *true*.
   - Then 9 seconds of idle time go by, and
   - *enabled* is set to *false*.

If *enabled* is set to *true* again, the *onTimer* will fire after another 10 seconds has gone by, even though there was only 1 second left before the timer was disabled.

The *interval* property can only be set to a value of zero or greater. The *interval* property may be a fraction of a second; the resolution of the timer is one system clock tick, approximately 0.055 seconds. When *interval* is zero, the timer fires once per clock tick.

Setting the *interval* property always resets the internal counter to the newly specified time.

MDY( )

Topic group Related topics

Returns a specified date as a character string in MONTH DD, YY format.

Syntax

MDY(<expD>)

<expD>

The date to return as a character string in MONTH DD, YY format.

Description

MDY( ) returns a date in MONTH DD, YY or MONTH DD, YYYY format, where MONTH is the full month name, DD is the day number, and YY is the year number. If SET CENTURY is OFF (the default), MDY( ) returns the year as 2 digits. If SET CENTURY is ON, MDY( ) returns the year as 4 digits. MDY( ) always returns the day portion as 2 digits, with a leading zero for the first nine days of the month.

If you pass an invalid date to MDY( ), it returns "Unknown 00, 00" or "Unknown 00, 0000".

MONTH( )

Topic group Related topics Example

Returns the number of the month for a specified date.

Syntax

MONTH(<expD>)

<expD>

The date whose corresponding month number you want to return.

Description

MONTH( ) returns a date's month number:

Month	Number
January	1
February	2
March	3

April	4
May	5
June	6
July	7
August	8
September	9
October	10
November	11
December	12

To return the name of the month instead of the number, use CMONTH( ). MONTH( ) returns zero for a blank date.

onTimer

Topic group Related topics Example

When the timer's interval has elapsed.

Parameters

none

Property of

Timer

Description

A Timer object's *onTimer* event is fired every time the amount of idle time specified by the timer's *interval* property has elapsed.

Like all event handlers, inside the *onTimer* event handler, the reference *this* refers to the Timer object itself. To refer to other objects, add references to those objects as properties to the Timer object before activating the timer.

While processing the *onTimer* event, all active timers are suspended, since *dBASE Plus* is busy processing code. Once the *onTimer* event handler has completed, its internal counter is reset to the *interval*, and all active timers resume counting.

If a Timer is intended to go off only once instead of repeatedly, set the *enabled* property to *false* in the *onTimer* event handler.

parse( )

Topic group Related topics Example

Returns time equivalent of a date/time string, in milliseconds.

Syntax

Date.*parse*(<date expC>)

<date expC>

The date/time string you want to convert.

Property of

Date

Description

The *parse*( ) method returns the number of milliseconds since January 1, 1970 00:00:00 GMT for the specified date/time string, defaulting to the operating system's current time zone setting. For example, if the time zone is currently set to United States Eastern Standard Time, which is five hours behind GMT, then Date.*parse*("Sep 14 1995 11:20") yields a time which is equivalent to 16:20 GMT.

The string may be in any of the forms acceptable to the Date class constructor, as described under class Date. In contrast, the UTC( ) method uses numeric parameters for each of the date and time components and assumes GMT as the time zone.

Because *parse*( ) is a static class method, you call it via the Date class, not a Date object.

SECONDS( )

Topic group Related topics

Returns the number of seconds that have elapsed on your computer's system clock since midnight.

Syntax

SECONDS( )

Description

SECONDS( ) returns the number of seconds to the hundredth of a second that have elapsed on your system clock since midnight (12 am). There are 86,400 seconds in a day, so the maximum value SECONDS( ) can return is 86,399.99, just before midnight.

Use SECONDS( ) to calculate the amount of time that portions of your program take to run. SECONDS( ) is more convenient for this purpose than TIME( ) because SECONDS( ) returns a number rather than a character string.

You can also use SECONDS( ) instead of ELAPSED( ) to determine elapsed time for the current day to within hundredths of a second.

SET CENTURY

Topic group Related topics

Controls the format in which *dBASE Plus* displays the year portion of dates.

Syntax

SET CENTURY on | off

Description

When SET CENTURY is ON, *dBASE Plus* displays dates in the current format with 4-digit years; when SET CENTURY is OFF, *dBASE Plus* displays dates in the current format with 2-digit years.

You can enter a date with a 2-, 3-, or 4-digit year whether SET CENTURY is ON or OFF. *dBASE Plus* assumes that 2-digit years are in the epoch designated by SET EPOCH, by default 1950. If SET CENTURY is OFF, *dBASE Plus* truncates any digits to the left of the last two when displaying the date. However, *dBASE Plus* stores the correct value of the date internally.

The following table shows the how *dBASE Plus* displays and stores dates depending on the setting of SET CENTURY. (The table assumes SET DATE is AMERICAN and SET EPOCH is 1950.)

As the table shows, SET CENTURY doesn't affect the relationship between how you enter a date and how *dBASE Plus* evaluates and stores it. SET CENTURY affects only how *dBASE Plus* displays the year portion of the date.

You enter date as	*dBASE Plus* stores date as YYYYMMDD	With SET CENTURY ON *dBASE Plus* displays	With SET CENTURY OFF *dBASE Plus* displays
{10/13/94}	19941013	10/13/1994	10/13/94
{10/13/994}	09941013	10/13/0994	10/13/94
{10/13/1994}	19941013	10/13/1994	10/13/94
{10/13/2094}	20941013	10/13/2094	10/13/94

SET DATE

Topic group Related topics

Specifies the format *dBASE Plus* uses for the display and entry of dates.

Syntax

SET DATE [TO]
AMERICAN | ANSI | BRITISH | FRENCH | GERMAN | ITALIAN | JAPAN | USA | MDY | DMY | YMD

TO

Included for readability only; TO has no affect on the operation of the command.

AMERICAN | ANSI | BRITISH | FRENCH | GERMAN | ITALIAN | JAPAN | USA | MDY | DMY | YMD

The options correspond to the following formats:

Option	Format
AMERICAN	MM/DD/YY
ANSI	YY.MM.DD
BRITISH	DD/MM/YY
FRENCH	DD/MM/YY
GERMAN	DD.MM.YY
ITALIAN	DD-MM-YY
JAPAN	YY/MM/DD
USA	MM-DD-YY
MDY	MM/DD/YY
DMY	DD/MM/YY
YMD	YY/MM/DD

Description

SET DATE determines how *dBASE Plus* displays dates; and how literal date strings, like those in curly braces ({ }), are interpreted. If SET CENTURY is ON, *dBASE Plus* displays all formats with a 4-digit year.

The default for SET DATE is set by the Regional Settings in the Windows Control Panel. To change the default, set the DATE parameter in PLUS.ini. To do so, either use the SET command to specify the setting interactively, or enter the DATE parameter directly in PLUS.ini.

SET DATE overrides any prior SET MARK setting. However, you can use SET MARK after SET DATE to change the date separator character.

SET DATE TO

Topic group Related topics

Sets the system date.

Syntax

SET DATE TO <expC>

<expC>

The character expression, in the current date format, to set as the current system date.

Description

Use SET DATE TO to reset the date on your system clock. The date string in <expC> must match the current setting of SET DATE.

The date must be in the range from January 1, 1980, to December 31, 2099.

SET EPOCH

Topic group Related topics

Sets the base year for interpreting two-digit years in dates.

Syntax

SET EPOCH TO <expN>

Default

The default base year is 1950, yielding years from 1950 to 2049.

Description

Use SET EPOCH to change how two-digit years are interpreted. This allows you to keep SET CENTURY OFF, while enabling entry of dates that cross a century boundary. The following table shows how dates are interpreted using three different SET EPOCH settings:

Date	1900	1930	2000
{5/5/00}	05/05/1900	05/05/2000	05/05/2000
{5/5/30}	05/05/1930	05/05/1930	05/05/2030
{5/5/99)	05/05/1999	05/05/1999	05/05/2099

For example, if you SET EPOCH TO 1930, you can continue to use most applications with two-digit years unchanged well into the 21st century, (although you would no longer be able to enter dates before 1930, which would not be a problem with many applications). If your applications use dates that span more than one hundred years, then SET EPOCH alone will not help; you must SET CENTURY ON.

The base year setting takes effect whenever dates are interpreted. In programs, two-digit years in literal dates are evaluated at compile-time. If you use SET EPOCH, be sure it is set correctly when you compile code or run new or changed programs.

SET EPOCH is session-based. You may get the value of SET EPOCH with the SET( ) and SETTO( ) functions.

SET HOURS

Topic group

Determines whether times are displayed in military format, or with an attached AM/PM indicator.

Syntax

SET HOURS TO [<expN>]

<expN>

The number 12 or 24.

Description

Setting SET HOURS to 12 displays time with an attached AM/PM indicator. Setting SET HOURS to 24 displays time in military format. SET HOURS TO (with no argument) restores the default setting.

SET MARK

Topic group Related topics

Determines the character *dBASE Plus* uses to separate the month, day, and year when it displays dates.

Syntax

SET MARK TO [<expC>]

<expC>

The single date separator character. You can specify more than one character for <expC>, but *dBASE Plus* uses only the first one.

Description

Use SET MARK to change the date separator from the default character. For example, if you issue SET DATE AMERICAN, the date separator character is a forward slash (/), and *dBASE Plus* displays dates in MM/DD/YY format. However, if you specify SET MARK TO "." after issuing SET DATE AMERICAN, *dBASE Plus* displays dates in the format MM.DD.YY. If you issue SET DATE AMERICAN again, the format returns to MM/DD/YY.

Issuing SET MARK TO without <expC> resets the date separator character to that of the current date format.

SET MARK controls the separator used for display only. You may use any valid separator character when designating a literal date.

SET TIME

Topic group Related topics

Sets the system time.

Syntax

SET TIME TO <expC>

<expC>

The time, which you must specify in one of the following formats:

HH
HH:MM or HH.MM
HH:MM:SS or HH.MM.SS

Description

Use SET TIME to reset your system's clock.

setDate( )

Topic group Related topics

Sets day of month.

Syntax

<oRef>.*setDate*(<expN>)

<oRef>

The Date object whose day you want to change.

<expN>

The day of month number, normally between 1 and 31.

Property of

Date

Description

The *setDate*( ) method sets the day of month for the Date object.

setHours( )

Topic group Related topics

Sets hours portion of time.

Syntax

<oRef>.*setHours*(<expN>)

<oRef>

The Date object whose hours you want to change.

<expN>

The hour number, normally between 0 and 23.

Property of

Date

Description

The *setHours*( ) method sets the hours portion of "time" for the Date object.

setMinutes( )

Topic group Related topics

Sets minutes portion of time.

Syntax

<oRef>.*setMinutes*(<expN>)

<oRef>

The Date object whose minutes you want to change.

<expN>

The minute number, normally between 0 and 59.

Property of

Date

Description

The *setMinutes*( ) method sets the minutes portion of "time" for the Date object.

setMonth( )

Topic group Related topics

Sets month of year.

Syntax

<oRef>.*setMonth*(<expN>)

<oRef>

The Date object whose month you want to change.

<expN>

The month number, normally between 0 and 11: 0 for January, 1 for February, and so on, up to 11 for December.

Property of

Date

Description

The *setMonth*( ) method sets the month of year for the Date object.

setSeconds( )

Topic group Related topics

Sets seconds portion of time.

Syntax

<oRef>.*setSeconds*(<expN>)

<oRef>

The Date object whose seconds you want to change.

<expN>

The number of seconds, normally between 0 and 59.

Property of

Date

Description

The *setSeconds*( ) method sets the seconds portion of "time" for the Date object.

setTime( )

Topic group Related topics

Sets date/time of Date object.

Syntax

<oRef>.setTime(<expN>)

<oRef>

The Date object whose time you want to set.

<expN>

The number of milliseconds since January 1, 1970 00:00:00 GMT for the desired date/time.

Property of

Date

Description

While you may use standard date/time nomenclature when creating a new Date object, the *setTime*( ) method requires a number of milliseconds. Therefore the *setTime*( ) method is used primarily to copy the date/time from one Date object to another. If you tried copying dates like this:

```
d1 = new Date( "Aug 24 1996" )
d2 = new Date( )
d2 = d1 // Copy date
```

what you're actually doing is copying an object reference for the first Date object into another variable. Both variables now point to the same object, so changing the date/time in one would appear to change the date/time in the other.

To actually copy the date/time, use the *setTime*( ) and *getTime*( ) methods:

```
d1 = new Date( "Aug 24 1996" )
d2 = new Date( )
d2.setTime( d1.getTime( ) ) // Copy date
```

If you're copying the date/time when you're creating the second Date object, you can use the millisecond value in the Date class constructor:

```
d1 = new Date( "Aug 24 1996" )
d2 = new Date( d1.getTime( ) ) // Create copy of date
```

You may also perform date math by adding or subtracting milliseconds from the value.

setYear( )

Topic group Related topics

Sets year of date.

Syntax

<oRef>.*setYear*(<expN>)

<oRef>

The Date object whose year you want to change.

<expN>

The year. For years in the range 1950 to 2049, you can specify the year as either a 2-digit or 4-digit year.

Property of

Date

Description

The *setYear*( ) method sets the year for the Date object.

TIME( )

Topic group Related topics

Returns the system time as a character string in HH:MM:SS or HH:MM:SS.hh format.

Syntax

TIME([<exp>])

<exp>

Any expression, which causes TIME( ) to return the current time to the hundredth of a second.

Description

TIME( ) returns a character expression that is your computer system's current time. If you do not pass TIME( ) an expression, it returns the current system time in HH:MM:SS format, where HH is the hour, MM the minutes, and SS the seconds.

If you pass TIME( ) an expression, it returns the current system time in HH:MM:SS.hh format, where .hh is hundredths of a second. The type and value of the expression you pass to TIME( ) has no effect other than to make it include hundredths of a second.

To change the system time, use SET TIME.

toGMTString( )

Topic group Related topics Example

Converts the date into a string, using Internet (GMT) conventions.

Syntax

<oRef>.*toGMTString*( )

<oRef>

The Date object you want to convert.

Property of

Date

Description

The *toGMTString*( ) method converts the date, as exists in the operating system's time zone setting, to GMT and returns a string such as, "Tue, 07 May 2002 02:55:27 GMT".

toLocaleString( )

Topic group Related topics Example

Converts the date into a string, using locale conventions.

Syntax

<oRef>.*toLocaleString*( )

<oRef>

The Date object you want to convert.

Property of

Date

Description

The *toLocaleString*( ) method converts the date to a string, using the display standards for the current locale setting, such as, "05/06/96 19:55:27".

dBASE Plus uses Windows' Regional settings from the Control Panel.

toString( )

Topic group Related topics Example

Converts the date into a string, using standard JavaScript conventions.

Syntax

<oRef>.*toString*( )

<oRef>

The Date or Number object you want to convert.

Property of

Date, Number

Description

The *toString*( ) method converts the date or Number Object Value to a string.

For the Date objects it returns in standard JavaScript format, which includes the complete time zone description. For example,

```
"Mon May 06 19:55:27 Pacific Daylight Time 2002"
```

TTIME( )

Topic group Related topics

Returns a value representing the current system time in the HH:MM:SS format.

Syntax

TTIME( )

Description

The TTIME( ) function returns a time value that is your computer systems current time. When "SET HOURS" is set to 12, this value is displayed with an attached AM/PM indicator.

TTIME( ) is quite similar to the TIME( ) function. However, while the TIME( ) function results in a military time character string, TTIME( ) results in a time value.

Since the actual value of TTIME( ) is measured in seconds, adding 60 to TTIME( ) is the equilvalent of adding 1 minute.

TTIME( ) values can be converted to character strings using the TTOC( ) function, and back to values using CTOT( ).

TTOC( )

Topic group Related topics

"Time to Character" converts a TTIME( ) value - your computer systems current time - to a literal string.

Syntax

TTOC(<tVar>)

<tVar>

A TTIME( ) variable or value

Description

Use TTOC( ) to convert a Time value into a literal Time string. "Time to Character" results in an HH:MM:SS format. When "SET HOURS" is set to 12, the TTOC string is displayed with an attached AM/PM indicator.

UTC( )

Topic group Related topics Example

Returns time equivalent of the specified date/time parameters using GMT, in milliseconds.

Syntax

Date.UTC(<year expN>, <month expN>, <day expN>
[, <hours expN> [, <minutes expN> [, <seconds expN>]]])

<year expN>

The year.

<month expN>

A number representing the month, between 0 and 11: zero for January, one for February, and so on, up to 11 for December.

<day expN>

The day of the month, from 1 to 31.

<hours expN>

The hours portion of the time, from 0 to 23.

<minutes expN>

The minutes portion of the time, from 0 to 59.

<seconds expN>

The seconds portion of the time, from 0 to 59.

Property of

Date

Description

UTC( ) returns the number of milliseconds since January 1, 1970 00:00:00 GMT, using the specified date/time parameters, and assumes GMT as the local time zone. In contrast, the *parse*( ) method takes a string as a parameter, and uses the operating system's current time zone setting as the default.

Because UTC( ) is a static class method, you call it via the Date class, not a Date object.

YEAR( )

Topic group Related topics

Returns the year of a specified date.

Syntax

YEAR(<expD>)

<expD>

The date whose corresponding year number you want to return.

Property of

Date

Description

YEAR( ) returns a date's 4-digit year number. The SET CENTURY setting has no effect on YEAR( ).

YEAR( ) returns zero for a blank date.

Math / Money

Class Number

Topic group Related topics

The NUMBER() object is used for High Precision Math calculations.

Syntax

[<oRef> =] new Number([<nNumber expN>])

<oRef>

A variable or property in which to store a reference to the newly created Number object.

<nNumber expN>

An optional number or numeric expression

Description

It allows for various properties and events to be used to help with very large numbers. This implements most of the specification for the IEEE 754 standard (http://speleotrove.com/decimal/)

The Number class contains a value property that can a single value at a time in high precision format. It has other properties for adjusting rounding, precision, maximum exponents, .

It has numerous mathematical methods for performing common operations on the value as well as some methods which use the value as input and return a result based on the value.

Setting the value:

You can set the value of a number object either at construction or anytime later as follows:

At construction:

n = new number(<somevalue>)

After construction:

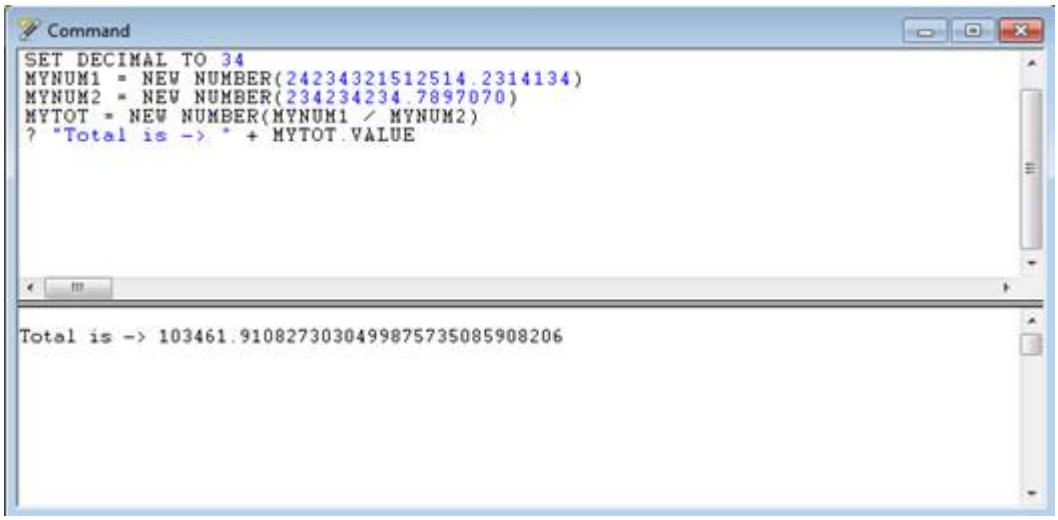
n.assign(<somevalue>)

or

n.value = <somevalue>

Example:

```
SET PRECISION TO 34
SET DECIMAL TO 34
MYNUM1 = NEW NUMBER(24234321512514.2314134)
MYNUM2 = NEW NUMBER(234234234.7897070)
MYTOT = NEW NUMBER(MYNUM1 / MYNUM2)
? “Total is -> “ + MYTOT.VALUE
```



Another key feature of the NUMBER object is that is has its own precision built into the implementation, which means the SET PRECISION setting does not apply to it. This is because he has own setting of precision and it is number of significand digits can be stored; a value in the range 1 through 999 999 999.

Database Storage:

Do to current limitations in both dBASE and also the BDE, a Number() object cannot be stored in a field in the database. There is no corresponding data type to represent that natively. In this case, if you are using dBASE or Paradox files, the value must be stored as a String data type.

Key in mind that some databases could support high precision math storage, please refer to the particular database documentation for further insights and details.

PROPERTIES

Property	Default	Description
baseClassName	NUMBER	The default baseClassName of the component
className	NUMBER	The user defined or default className of the component
clamp	False	The clamp field controls explicit exponent clamping, as is app formats. When False, a result exponent is limited to a maximum of em result will be clamped to be in this range). When True, a result exponent has the same minimum but is li zeros, this may cause the coefficient of a result to be padded range. For example, if emax is +96 and digits is 7, the result 1.23E+ clamp were False, but would give [0, 1230000, 90] if clamp w In addition, when True, clamp limits the length of NaN payloa conversion from a string.
digits	1	The digits field is used to set the precision to be used for an o length if necessary. The valid range must have a value in the This cannot be set any higher than the Precision number.
emax	999999999	The emax field is used to set the magnitude of the largest adj calculated as though the number were expressed in scientific before the decimal point). If the adjusted exponent for a result or conversion would be l The valid range must have a value in the range 0 through 999
emin	-999999999	The emin field is used to set the smallest adjusted exponent t calculated as though the number were expressed in scientific before the decimal point). If the adjusted exponent for a result or conversion would be s also inexact, an underflow results. The exponent of the small digits+1. emin is usually set to –emax or to –(emax–1). emin is of type int32_t, and must have a value in the range –9
exponent	0	The exponent field holds the exponent of the number. Its ran exponent of the number be balanced and fit within a whole nu through +999,999,999). The adjusted exponent is the expone digit before the decimal point, and is therefore given by expon When the extended flag in the context is 1, gradual underflow limit for the adjusted exponent becomes –999,999,999–(preci the adjusted exponent may then have 10 digits.
level	1 - ANSI	This refers to either IEEE or ANSI calculation method.
mantissa	0	The use of mantissa to refer to significant digits in a floating p mantissa would represent 0.14159
precision	34	Determines the number of digits used when comparing numb
roundType	0 – Half Even	The round field is used to select the rounding algorithm to be one of the values in the rounding enumeration: • 0 – Half Even; round to nearest; if equidistant, round so • 1 – Half Up; round to nearest; if equidistant, round up. • 2 – Half Down; round to nearest; if equidistant, round do • 3 – Up; round away from 0. • 4 – Down; round towards 0 (truncation). • 5 – Ceiling; round towards +Infinity.

		• 6 – Floor; round towards –Infinity. • 7 – Zero Five; the same as UP, except that rounding up Overflow the result is the same as for DOWN. • 8 – None; The same as HALF EVEN.
signed	false	Having both positive and negative varieties. To be signed wo
state	0 - Zero	represents the state of a number
status	0 - None	The status field comprises one bit for each of the exceptional remains set until cleared by the user, so more than one condi status is of type unsigned integer. Bits in the field must only b library modules when exceptional conditions occur, but are ne appropriate (for example, after handling the exceptional condi
value	0	This is the value established once a number has been assign

EVENTS

No Events

METHODS

This following is how a method would be called:

```
mytotal = new number()
mytot1al.value := 103461.91082730304998757350859082O7
? mytotal.toString()
? mytotal.tonumber()
? mytotal.sqrt()
```

NOTE: *Calling methods do not change the value property. They return a result based on the value property.*
However, some methods DO update the value property:
add()
sub()
mul()
div()

Method Name	Parameters	Functionality
abs	None	Returns the absolute value of a specified number.
acos	None	Returns the inverse cosine (arccosine) of a number (in radians)

add	<Number>	Addition.
		Adds <Number> to current Number Object.
		
		Example:
		newnum = new number()
		newnum.value = 10 //value would be 10
		? newnum.add(10) //add 10 to the value
		? newnum.value
		
		Output:
		20
		10
		
		Remember the methods do not change the value of the number object; it does use it for the calculation.

asin	None	Returns the inverse sine (arcsine) of a number (in radians)

atan	None	Returns the inverse tangent (arctangent) of a number (in radians)
assign	<nExp>	Assigns a new numeric value
cos	None	Returns the trigonometric cosine of an angle (in radians)
degree		Returns the degrees from the angles measured in radians. **Example:** set precision to 34 set decimal to 34 num1 = new number(.5) ? num1.degree() Output: 28.647889756541160438399077407052 58

div	<Number>	Division by <Number>
		
		Example:
		newnum = new number()
		newnum.value = 10 //value would be 10
		? newnum.div(2) //divides newnum by 2
		? newnum.value
		
		Output:
		5
		10
		
		Remember the methods do not change the value of the number object; it does use it for the calculation.

exp	None	Returns **e** raised to a specified power.
isFinite	None	This function is used to test whether a number is finite. Returns 1 (true) if the number is finite, or 0 (false) otherwise (that is, it is an infinity or a NaN).
isFloat	None	This function is used to test whether a number is a Float. Returns 1 (true) if number is a Float or 0 (false) otherwise.
isInf	None	This function is used to test whether a number is infinite. Returns 1 (true) if the number is infinite, or 0 (false) otherwise (that is, it is a finite number or a NaN).
isInt	None	This function is used to test whether a number is an Integer. Returns 1 (true) if number is an Integer or 0 (false) otherwise.
isNaN	None	This function is used to test whether a number is a NaN (**N**ot **a N**umber). Returns 1 (true) if the number is a NaN, or 0 (false) otherwise.
isNeg	None	This function is used to test whether a number is negative (either minus zero, less than zero, or a NaN with a sign of 1). Returns 1 (true) if the number is negative or 0 (false) otherwise.
isZero	None	This function is used to test whether a number is a zero (either positive or negative). Returns 1 (true) if the number is zero, or 0 (false) otherwise.
log	None	Returns the logarithm to the base e (natural logarithm) of a specified number.
log10	None	Returns the logarithm to the base 10 of a specified number.
logB	None	Returns a floating-point value extracted exponent from the internal floating-point representation of x.
mod	None	Returns the modulus (remainder) of one number divided by another.

mul	<nMultiplier>	Multiplication
		
		Example:
		newnum = new number()
		newnum.value = 10 //value would be 10
		? newnum.mul(10) //Multiply by 10

		? newnum.value Output: 100 10 Remember the methods do not change the value of the number object; it does use it for the calculation.
negate		
pi	None	Returns the approximate value of pi, the ratio of a circle's circumference to its diameter.
pow		Exponentiation is a mathematical operation, written as **b**^{n}, involving two numbers, the base **b** and the exponent (or index or power) **n**.
radian	None	Returns the Radians calculated form the NUMBER.VALUE **Example:** set precision to 34 set decimal to 34 num1 = new number(34) ? num1.radian() Output: 0.59341194567807205615405486128612831283

round	<nRoundDecimalPlace>	Sets the number object to a new number rounded to the nearest integer or a specified number of decimal places.
		n = new number(1.56)
		n.roundtype = 6 //floor
		n.round(1)
		?n // returns 1.6

sin	None	Returns the trigonometric sine of an angle (in radians)
sqrt	None	Returns the square root of a number.

sub	<nSubtractNumber>	Subtraction
		
		Example:
		newnum = new number()
		newnum.value = 10 //value would be 10
		? newnum.sub(10) //Subtract 10 from the value
		? newnum.value
		
		Output:
		0
		10
		
		Remember the methods do not change the value of the number object; it does use it for the calculation.

tan	None	Returns the trigonometric tangent of an angle (in radians)
toNumber	None	Returns the number from the object.
toSting	None	Returns the string representation of the value properties of the object.

Class NumberException

Topic group Related topics

Represents an object that describes an numberException condition.

Syntax

[<oRef> =] new NumberException( )

<oRef>

A variable or property in which to store a reference to the newly created NumberException object.

Description

Represents an object that describes an numberException condition.

PROPERTIES

Property	Default	Description
baseClassName	NUMBEREXCEPTION	Identifies the object as an instance of the NumberException c
className	NUMBEREXCEPTION	Identifies the object as an instance of a custom class. When
code	0	A numeric code to identify the type of exception.
fileName	<empty string>	The name of the file in which a system-generated exception
lineNo	0	The line number in the file in which a system-generated excep

Math / Money

Topic group

dBASE Plus supports a wide range of mathematic, trigonometric, and financial functions.

Starting with dBASE PLUS 8, a new High Precision Math (HPM) library, and wrapper has been added to the product. This new library implements the IEEE 754 Standard for Floating Point Arithmetic.

> **Note:** More information on the IEEE 754 standard can be found at the following web address: http://en.wikipedia.org/wiki/IEEE_754-2008 To quote Wikipedia on the subject: "*The IEEE Standard for Floating-Point Arithmetic (IEEE 754) is a technical standard for floating-point computation established in 1985 by the Institute of Electrical and Electronics Engineers (IEEE). Many hardware floating point units use the IEEE 754 standard. The current version, IEEE 754-2008 published in August 2008, includes nearly all of the original IEEE 754-1985 standard and the IEEE Standard for Radix-Independent Floating-Point Arithmetic (IEEE 854-1987). The international standard ISO/IEC/IEEE*

60559:2011 (with identical content to IEEE 754) has been approved for adoption through JTC1/SC 25 under the ISO/IEEE PSDO Agreement and published."

dBASE wraps this standard around our number handling interface, which supports the following concepts:

- Arithmetic formats: sets of binary and decimal floating-point data, which consist of finite numbers (including signed zeros and subnormal numbers), infinities, and special "not a number" values (NaNs)
- Interchange formats: encodings (bit strings) that may be used to exchange floating-point data in an efficient and compact form
- Rounding rules: properties to be satisfied when rounding numbers during arithmetic and conversions
- Operations: arithmetic and other operations on arithmetic formats
- Exception handling: indications of exceptional conditions (such as division by zero, overflow, etc.)

When using a decimal floating point format the decimal representation will be preserved using:

- 7 decimal digits for decimal32
- 16 decimal digits for decimal64
- 34 decimal digits for decimal128

The main areas in dBASE you want to review include:

- The new NUMBER Object included in dBASE
- SET PRECISION TO...
- SET DECIMAL TO ...

ABS( )

Topic group Related topics

Returns the absolute value of a specified number.

Syntax

ABS(<expN>)

<expN>

The number whose absolute value you want to return.

Description

ABS( ) returns the absolute value of a number. The absolute value of a number represents its magnitude. Magnitude is always expressed as a positive value, so the absolute value of a negative number is its positive equivalent.

ACOS( )

Topic group Related topics

Returns the inverse cosine (arccosine) of a number.

Syntax

ACOS(<expN>)

<expN>

The cosine of an angle, from –1 to +1.

Description

ACOS( ) returns the radian value of the angle whose cosine is <expN>. ACOS( ) returns a number from 0 to pi radians. ACOS( ) returns zero when <expN> is 1. For values of x from 0 to pi, ACOS(y) returns x if COS(x) returns y.

To convert the returned radian value to degrees, use RTOD( ). For example, if the default number of decimal places is 2, ACOS(.5) returns 1.05 radians while RTOD(ACOS(.5)) returns 60.00 degrees.

Use SET DECIMALS to set the number of decimal places ACOS( ) displays.

To find the arcsecant of a value, use the arccosine of 1 divided by the value. For example, the arcsecant of 2 is ACOS(1/2), or 1.05 radians.

ASIN( )

Topic group Related topics

Returns the inverse sine (arcsine) of a number.

Syntax

ASIN(<expN>)

<expN>

The sine of an angle, from –1 to +1.

Description

ASIN( ) returns the radian value of the angle whose sine is <expN>. ASIN( ) returns a number from –pi/2 to pi/2 radians. ASIN( ) returns zero when <expN> is 0. For values of x from –pi/2 to pi/2, ASIN(y) returns x if SIN(x) returns y.

To convert the returned radian value to degrees, use RTOD( ). For example, if the default number of decimal places is 2, ASIN(.5) returns .52 radians while RTOD(ASIN(.5)) returns 30.00 degrees.

Use SET DECIMALS to set the number of decimal places ASIN( ) displays.

To find the arccosecant of a value, use the arcsine of 1 divided by the value. For example, the arccosecant of 1.54 is ASIN(1/1.54), or .71 radians.

ATAN( )

Topic group Related topics

Returns the inverse tangent (arctangent) of a number.

Syntax

ATAN(<expN>)

<expN>

Any positive or negative number representing the tangent of an angle.

Description

ATAN( ) returns the radian value of the angle whose tangent is <expN>. ATAN( ) returns a number from –pi/2 to pi/2 radians. ATAN( ) returns 0 when <expN> is 0. For values of x from –pi/2 to pi/2, ATAN(y) returns x if TAN(x) returns y.

To convert the returned radian value to degrees, use RTOD( ). For example, if the default number of decimal places is 2, ATAN(1) returns 0.79 radians, while RTOD(ATAN(1)) returns 45.00 degrees.

Use SET DECIMALS to set the number of decimal places ATAN( ) displays.

ATAN( ) differs from ATN2( ) in that ATAN( ) takes the tangent as the argument, but ATN2( ) takes the sine and cosine as the arguments.

To find the arccotangent of a value, subtract the arctangent of the value from pi/2. For example, the arccotangent of 1.73 is PI/2 – ATAN(1.73), or .52.

ATN2( )

Topic group Related topics

Returns the inverse tangent (arctangent) of a given point.

Syntax

ATN2(<sine expN>, <cosine expN>)

<sine expN>

The sine of an angle. If <sine expN> is 0, <cosine expN> can't also be 0.

<cosine expN>

The cosine of an angle. If <cosine expN> is 0, <sine expN> can't also be 0. When <cosine expN> is 0 and <sine expN> is a positive or negative (nonzero) number, ATN2( ) returns +pi/2 or –pi/2, respectively.

Description

ATN2( ) returns the angle size in radians when you specify the sine and cosine of the angle. ATN2( ) returns a number from –pi to +pi radians. ATN2( ) returns 0 when <sine expN> is 0. When you specify 0 for both arguments, *dBASE Plus* returns an error.

To convert the returned radian value to degrees, use RTOD( ). For example, if the default number of decimal places is 2, ATN2(1,0) returns 1.57 radians while RTOD(ATN2(1,0)) returns 90.00 degrees.

Use SET DECIMALS to set the number of decimal places ATN2( ) displays.

ATN2( ) differs from ATAN( ) in that ATN2( ) takes the sine and cosine as the arguments, but ATAN( ) takes the tangent as the argument. See ATAN( ) for instructions on finding the arccotangent.

CEILING( )

Topic group Related topics

Returns the nearest integer that is greater than or equal to a specified number.

Syntax

CEILING(<expN>)

<expN>

A number, from which to determine and return the integer that is greater than or equal to it.

Description

CEILING( ) returns the nearest integer that is greater than or equal to <expN>; in effect, rounding positive numbers up and negative numbers down towards zero. If you pass a number with any digits other than 0 as decimal digits, CEILING( ) returns the nearest integer that is greater than the number. If you pass an integer to CEILING( ), or a number with only 0s for decimal digits, it returns that number.

For example, if the default number of decimal places is 2,

CEILING(2.10) returns 3.00
CEILING(–2.10) returns –2.00
CEILING(2.00) returns 2.00
CEILING(2) returns 2
CEILING(–2.00) returns –2.00

Use SET DECIMALS to set the number of decimal places CEILING( ) displays.

See the table in the description of INT( ) that compares INT( ), FLOOR( ), CEILING( ), and ROUND( ).

COS( )

Topic group Related topics

Returns the trigonometric cosine of an angle.

Syntax

COS(<expN>)

<expN>

The size of the angle in radians. To convert an angle's degree value to radians, use DTOR( ). For example, to find the cosine of a 30-degree angle, use COS(DTOR(30)).

Description

COS( ) calculates the ratio between the side adjacent to an angle and the hypotenuse in a right triangle. COS( ) returns a number from –1 to +1. COS( ) returns 0 when <expN> is pi/2 or 3*pi/2 radians.

Use SET DECIMALS to set the number of decimal places COS( ) displays.

The secant of an angle is the reciprocal of the cosine of the angle. To return the secant of an angle, use 1/COS( ).

DTOR( )

Topic group Related topics

Returns the radian value of an angle whose measurement is given in degrees.

Syntax

DTOR(<expN>)

<expN>

A negative or positive number that is the size of the angle in degrees.

Description

DTOR( ) converts the measurement of an angle from degrees to radians. To convert degrees to radians, *dBASE Plus*;

Multiplies the number of degrees by pi
Divides the result by 180
Returns the quotient

A 180-degree angle is equivalent to pi radians.

Use DTOR( ) in the trigonometric functions SIN( ), COS( ), and TAN( ) because these functions require the angle value in radians. For example, to find the sine of a 45-degree angle, use SIN(DTOR(45)), which returns .71 if the default number of decimal places is 2.

Use SET DECIMALS to set the number of decimal places DTOR( ) displays.

EXP( )

Topic group Related topics

Returns e raised to a specified power.

Syntax

EXP(<expN>)

<expN>

The positive, negative, or zero power (exponent) to raise the number e to.

Description

EXP( ) returns a number equal to e (the base of the natural logarithm) raised to the <expN> power. For example, EXP(2) returns 7.39 because e^2 = 7.39.

EXP( ) is the inverse of LOG( ). In other words, if y = EXP(x), then LOG(y) = x.

Use SET DECIMALS to set the number of decimal places EXP( ) displays.

FLOOR( )

Topic group Related topics

Returns the nearest integer that is less than or equal to a specified number.

Syntax

FLOOR(<expN>)

<expN>

A number from which to determine and return the integer that is less than or equal to it.

Description

FLOOR( ) returns the nearest integer that is less than or equal to <expN>; in effect, rounding positive numbers down and negative numbers up away from zero. If you pass a number with any digits other than zero (0) as decimal digits, FLOOR( ) returns the nearest integer that is less than the number. If you pass an integer to FLOOR( ), or a number with only zeros for decimal digits, it returns that number.

For example, if the default number of decimal places is 2,

FLOOR(2.10) returns 2.00
FLOOR(–2.10) returns –3.00
FLOOR(2.00) returns 2.00
FLOOR(2) returns 2
FLOOR(–2.00) returns –2.00

Use SET DECIMALS to set the number of decimal places FLOOR( ) displays.

When you pass a positive number to it, FLOOR( ) operates exactly like INT( ). See the table in the description of INT( ) that compares INT( ), FLOOR( ), CEILING( ), and ROUND( ).

FV( )

Topic group Related topics

Returns the future value of an investment.

Syntax

FV(<payment expN>, <interest expN>, <term expN>)

<payment expN>

The amount of the periodic payment. Specify the payment in the same time increment as the interest and term. The payment can be negative or positive.

<interest expN>

The interest rate per period expressed as a positive decimal number. Specify the interest rate in the same time increment as the payment and term.

<term expN>

The number of payments. Specify the term in the same time increment as the payment and interest.

Description

Use FV( ) to calculate the amount realized (future value) after equal periodic payments (deposits) at a fixed interest rate. FV( ) returns a float representing the total of the payments plus the interest generated and compounded.

Express the interest rate as a decimal. For example, if the annual interest rate is 9.5%, <interest expN> is .095 (9.5/100) for payments made annually.

Express <payment expN>, <interest expN>, and <term expN> in the same time increment. For example, if the payment is monthly, express the interest rate per month, and the number of payments in months. You would express an annual interest rate of 9.5%, for example, as .095/12, which is 9.5/100 divided by 12 months.

The formula *dBASE Plus* uses to calculate FV( ) is as follows:

$$fv = pmt * \frac{1 + int^{term} - 1}{int}$$

where int = rate / 100

For the future value an investment of $350 made monthly for five years, earning 9% interest, the formula expressed as a dBL expression looks like this:

```
? FV(350,.09/12,60) // Returns 26398.45
? 350*((1+.09/12)^60-1)/(.09/12) // Returns 26398.45
```

In other words, if you invest $350/month for the next five years into an account that pays an annual interest rate of 9%, at the end of five years you will have $26398.45.

Use SET DECIMALS to set the number of decimal places FV( ) displays.

INT( )

Topic group Related topics

Returns the integer portion of a specified number.

Syntax

INT(<expN>)

<expN>

A number whose integer value you want to determine and return.

Description

Use INT( ) to remove the decimal digits of a number and retain only the integer portion, the whole number.

If you pass a number with decimal places to a function, command, or method that uses an integer as an argument, such as SET EPOCH, *dBASE Plus* automatically truncates that number, in which case you don't need to use INT( ).

The following table compares INT( ), FLOOR( ), CEILING( ), and ROUND( ). (In these examples, the value of the second ROUND( ) argument is 0.)

<expN>	INT( )	FLOOR( )	CEILING( )	ROUND( )
2.56	2	2	3	3
–2.56	–2	–3	–2	–3
2.45	2	2	3	2
–2.45	–2	–3	–2	–2

LOG( )

Topic group Related topics

Returns the logarithm to the base e (natural logarithm) of a specified number.

Syntax

LOG(<expN>)

<expN>

A positive nonzero number that equals e raised to the log. If you specify 0 or a negative number for <expN>, *dBASE Plus* generates an error.

Description

LOG( ) returns the natural logarithm of <expN>. The natural logarithm is the power (exponent) to which you raise the mathematical constant e to get <expN>. For example, LOG(5) returns 1.61 because e^1.61= 5.

LOG( ) is the inverse of EXP( ). In other words, if LOG(y) = x, then y = EXP(x).

Use SET DECIMALS to set the number of decimal places LOG( ) displays.

LOG10( )

Topic group Related topics

Returns the logarithm to the base 10 of a specified number.

Syntax

LOG10(<expN>)

<expN>

A positive nonzero number which equals 10 raised to the log. If you specify 0 or a negative number for <expN>, *dBASE Plus* returns an error.

Description

LOG10( ) returns the common logarithm of <expN>. The common logarithm is the power (exponent) to which you raise 10 to get <expN>. For example, LOG10(100) returns 2 because 10^2=100.

Use SET DECIMALS to set the number of decimal places LOG10( ) displays.

MAX( )

Topic group Related topics

Compares two numbers (or two date, character, or logical expressions) and returns the greater value.

Syntax

MAX(<exp1>, <exp2>)

<exp1>

A numeric, date, character, or logical expression to compare to a second expression of the same type.

<exp2>

The second expression to compare to <exp1>.

Description

Use MAX( ) to compare two numbers to determine the greater of the two values. You can use MAX( ) to ensure that a number is not less than a particular limit.

MAX( ) may also be used to compare two dates, character strings, or logical values, in which case MAX( ) returns:

- The later of the two dates. In *dBASE Plus*, a blank date is considered later than a non-blank date.
- The character string with the higher collation value. Collation values are determined by the language driver in use, and are case-sensitive. For example, with the DB437US driver, the letter "B" is higher than the letter "A", but "a" is higher than "B" (all lowercase letters are collated higher than uppercase letters).
- *true* if one or both logical expressions evaluate to *true*. (The logical OR operator has the same effect.)

If <exp1> and <exp2> are equal, MAX( ) returns their value.

MIN( )

Topic group Related topics

Compares two numbers (or two date, character, or logical expressions) and returns the lesser value.

Syntax

MIN(<exp1>, <exp2>)

<exp1>

A numeric, date, character, or logical expression to compare to a second expression of the same type.

<exp2>

The second expression to compare to <exp1>.

Description

Use MIN( ) to compare two numbers to determine the lesser of the two values. You can use MIN( ) to ensure that a number is not greater than a particular limit.

MIN( ) may also be used to compare two dates, character strings, or logical values, in which case MIN( ) returns:

- The earlier of the two dates. In *dBASE Plus*, a non-blank date is considered earlier than a blank date.
- The character string with the lower collation value. Collation values are determined by the language driver in use, and are case-sensitive. For example, with the DB437US driver, the letter "a" is lower than the letter "b", but "B" is lower than "a" (all uppercase letters are collated lower than lowercase letters).
- *false* if one or both logical expressions evaluate to *false*. (The logical AND operator has the same effect.)

If <exp1> and <exp2> are equal, MIN( ) returns their value.

MOD( )

Topic group Related topics

Returns the modulus (remainder) of one number divided by another.

Syntax

MOD(<dividend expN>, <divisor expN>)

<dividend expN>

The number to be divided.

<divisor expN>

The number to divide by.

Description

MOD( ) divides <dividend expN> by <divisor expN> and returns the remainder. In other words, MOD(X,Y) returns the remainder of x/y.

The modulus formula is

```
<dividend>-INT(<dividend>/<divisor>)*<divisor>
```

where INT( ) truncates a number to its integer portion.

Note

Earlier versions of dBASE used FLOOR( ) instead of INT( ) in the modulus calculation. This change only affects the result if <dividend expN> and <divisor expN> are not the same sign, which in itself is an ambiguous case.

The % symbol is also used as the modulus operator. It performs the same function as MOD( ). For example, the following two expressions are identical:

```
mod( x, 2 )
x % 2
```

PAYMENT( )

Topic group Related topics

Returns the periodic amount required to repay a debt.

Syntax

PAYMENT(<principal expN>, <interest expN>, <term expN>)

<principal expN>

The original amount to be repaid over time.

<interest expN>

The interest rate per period expressed as a positive decimal number. Specify the interest rate in the same time increment as the term.

<term expN>

The number of payments. Specify the term in the same time increment as the interest.

Description

Use PAYMENT( ) to calculate the periodic amount (payment) required to repay a loan or investment of <principal expN> amount in <term expN> payments. PAYMENT( ) returns a number based on a fixed interest rate compounding over a fixed length of time.

If <principal expN> is positive, PAYMENT( ) returns a positive number.
If <principal expN> is negative, PAYMENT( ) returns a negative number.

Express the interest rate as a decimal. For example, if the annual interest rate is 9.5%, <interest expN> is .095 (9.5/100) for payments made annually.

Express <interest expN> and <term expN> in the same time increment. For example, if the payments are monthly, express the interest rate per month, and the number of payments in months. You would express an annual interest rate of 9.5%, for example, as .095/12, which is 9.5/100 divided by 12 months.

The formula *dBASE Plus* uses to calculate PAYMENT( ) is as follows:

$$pmt = princ * \frac{int * (1 + int)^{term}}{(1 + int)^{term} - 1}$$

where int = rate/100

For the monthly payment required to repay a principal amount of $16860.68 in five years, at 9% interest, the formula expressed as a dBL expression looks like this:

```
? PAYMENT(16860.68,.09/12,60) // Returns 350.00
nTemp = (1 + .09/12)^60
? 16860.68*(.09/12*nTemp)/(nTemp-1) // Returns 350.00
```

Use SET DECIMALS to set the number of decimal places PAYMENT( ) displays.

PI( )

Topic group Related topics

Returns the approximate value of pi, the ratio of a circle's circumference to its diameter.

Syntax

PI( )

Description

PI( ) returns a number that is approximately 3.141592653589793. pi is a constant that can be used in mathematical calculations. For example, use it to calculate the area and circumference of a circle or the volume of a cone or cylinder.

Use SET DECIMALS to set the number of decimal places PI( ) displays.

PV( )

Topic group Related topics

Returns the present value of an investment.

Syntax

PV(<payment expN>, <interest expN>, <term expN>)

<payment expN>

The amount of the periodic payment. Specify the payment in the same time increment as the interest and term. The payment can be negative or positive.

<interest expN>

The interest rate per period expressed as a positive decimal number. Specify the interest rate in the same time increment as the payment and term.

<term expN>

The number of payments. Specify the term in the same time increment as the payment and interest.

Description

PV( ) is a financial function that calculates the original principal balance (present value) of an investment. PV( ) returns a float that is the amount to be repaid with equal periodic payments at a fixed interest rate compounding over a fixed length of time. For example, use PV( ) if you want to know how much you need to invest now to receive regular payments for a specified length of time.

Express the interest rate as a decimal. For example, if the annual interest rate is 9.5%, <interest expN> is .095 (9.5 / 100) for payments made annually.

Express <payment expN>, <interest expN>, and <term expN> in the same time increment. For example, if the payment is monthly, express the interest rate per month, and the number of

payments in months. Express an annual interest rate of 9.5%, for example, as .095/12, which is 9.5/100 divided by 12 months.

The formula *dBASE Plus* uses to calculate PV( ) is as follows:

$$pv = pmt * \frac{(1 + int)^{term} - 1}{int * (1 + int)^{term}}$$

where int = rate 100

For the present value of an investment earning 9% interest, to be paid at $350 monthly for five years, the formula expressed as a dBL expression looks like this:

```
? PV(350,.09/12,60) // Returns 16860.68
nTemp = (1 + .09/12)^60
? 350*(nTemp-1)/(.09/12*nTemp) // Returns 16860.68
```

In other words, you have to invest $16,860.68 now into an account paying an interest rate of 9% annually to receive $350/month for the next five years.

Use SET DECIMALS to set the number of decimal places PV( ) displays.

RANDOM( )

Topic group Related topics

Returns a pseudo-random number between 0 and 1 exclusive (never 0 and never 1). *Same as RAND*( )

Syntax

RANDOM([<expN>])

<expN>

The number with which you want to seed RANDOM( ).

Description

Computers cannot generate truly random numbers, but you can use RANDOM( ) to generate a series of numbers that appear to have a random distribution. A series of pseudo-random numbers relies on a seed value, which determines the exact numbers that appear in the series. If you use the same seed value, you get the same series of numbers.

Pseudo-random numbers, when considered as a whole series, appear to be random; that is, you cannot tell from one number what the next will be. But the first number in the series is related to the seed value. Therefore, you should seed RANDOM( ) only once at the beginning of each series, like before simulating a card shuffle or randomly assigning work shifts. Seeding during a series defeats the design of the random number generator.

If you specify a positive <expN> value, RANDOM( ) uses that <expN> as the seed value, so a positive value should be used for testing, since the numbers will be the same each time. If <expN> is negative, RANDOM( ) uses a seed value based on the number of seconds past midnight on your computer system clock. As a result, a negative <expN> value most likely will give you a different series of random numbers each time.

If you don't specify <expN>, or use zero, RANDOM( ) returns the next number in the series.

When *dBASE Plus* first starts up, the random number generator is seeded with a fixed internal seed value of 179757.

Use SET DECIMALS to set the number of decimal places RANDOM( ) displays.

ROUND( )

Topic group Related topics

Returns a specified number rounded to the nearest integer or a specified number of decimal places.

Syntax

ROUND(<expN 1> , <expN 2>)

<expN 1>

The number you want to round.

<expN 2>

If <expN 2> is positive, the number of decimal places to round <expN 1> to. If <expN 2> is negative, whether to round <expN 1> to the nearest tens, hundreds, thousands, and so on.

Description

Use ROUND( ) to round a number to a specified number of decimal places or to a specified tens, hundreds, thousands value, and so forth. Use ROUND( ) with SET DECIMALS to round a number and remove trailing zeros.

If the digit in position <expN 2> + 1 is between 0 and 4 inclusive, <expN 1> (with <expN 2> decimal places) remains the same; if the digit in position <expN 2> + 1 is between 5 and 9 inclusive, the digit in position <expN 2> is increased by 1.

Use 0 as <expN 2> to round a number to the nearest whole number. Using –1 rounds a number to the nearest multiple of ten; rounding to a –2 rounds a number to the nearest multiple of one hundred; and so on. For example, ROUND(14932,–2) returns 14900 and ROUND(14932,–3) returns 15000.

For example, if the default number of decimal places is 2,

ROUND(2.50,0) returns 3.00
ROUND(–2.50,0) returns –2.00
ROUND(2.00,0) returns 2.00
See the table in the description of INT( ) that compares INT( ), FLOOR( ), CEILING( ), and ROUND( ).

RTOD( )

Topic group Related topics

Returns the degree value of an angle measured in radians.

Syntax

RTOD(<expN>)

<expN>

A negative or positive number that is the size of the angle in radians.

Description

RTOD( ) converts the measurement of an angle from radians to degrees.

To convert radians to degrees, *dBASE Plus*

Multiplies the number of radians by 180
Divides the result by pi
Returns the quotient

An angle of pi radians is equivalent to 180 degrees.

Use RTOD( ) with the trigonometric functions ACOS( ), ASIN( ), ATAN( ), and ATN2( ) to convert the radian return values of these functions to degrees. For example, if the default number of decimal places is 2, ATAN(1) returns the value of the angle in radians, 0.79, while RTOD(ATAN(1)) returns the value of the angle in degrees, 45.00.

Use SET DECIMALS to set the number of decimal places RTOD( ) displays.

SET CURRENCY

Topic group Related topics

SET CURRENCY determines the character(s) used as the currency symbol, and the position of that symbol when displaying monetary values

Syntax

SET CURRENCY left | right

SET CURRENCY TO [<expC>]

LEFT

Places currency symbol(s) to the left of currency numbers.

RIGHT

Places currency symbol(s) to the right of currency numbers.

<expC>

The characters that appear as a currency symbol. Although dBL imposes no limit to the length of <expC>, it recognizes only the first nine characters. You can't include numbers in <expC>.

Description

Currency symbols are displayed for numbers when you use the "$" template symbol in a formatting template or the TRANSFORM( ) function. The defaults for SET CURRENCY are set by the Regional settings of the Windows Control Panel.

Use SET CURRENCY left | right to specify the position of currency symbol(s) in monetary numeric values. Use SET CURRENCY TO to establish a currency symbol other than the default.

When SET CURRENCY is LEFT, *dBASE Plus* displays only as many currency symbols as fit, together with the digits to the left of any decimal point, within ten character spaces.

SET CURRENCY TO without the <expC> option resets the currency symbol to the default set with the Regional settings of the Windows Control Panel.

SET DECIMALS

Topic group Related topics

Determines the number of decimal places of numbers to display.

Syntax

SET DECIMALS TO [<expN>]

<expN>

The number of decimals places, from 0 to 34. The default is 2.

Description

Use SET DECIMALS to specify the number of decimal places of numbers you want *dBASE Plus* to display. SET DECIMALS affects the display of most mathematical calculations, but not the way numbers are stored on disk or maintained internally.

Excess digits are rounded when a number is displayed. For example, with the default setting of two decimal places, the number 1.995 is displayed as 2.00.

Use SET PRECISION to set the number of decimal places used in comparisons. SET DECIMALS and SET PRECISION are independent settings.

SET DECIMALS TO without <expN> resets the number of decimal places back to the default of 2.

SET POINT

Topic group Related topics

Specifies the character that separates decimal digits from integer digits in numeric display.

Syntax

SET POINT TO [<expC>]

<expC>

The character representing the decimal point. You can specify more than one character, but *dBASE Plus* uses only the first one. If you specify a number as a character for <expC> (for example, "3"), *dBASE Plus* returns an error.

The default is set by the Regional Settings of the Windows Control Panel.

Description

SET POINT affects both numeric input and display with commands such as EDIT. SET POINT also affects numeric display with commands such as DISPLAY MEMORY, STORE, =, and the PICTURE "." template character. You must use the period in the PICTURE option, regardless of the setting of SET POINT.

SET POINT has no effect on the representation of numbers in dBL expressions and statements. Only a period is valid as a decimal point. For example, if you SET POINT TO "," (comma) and issue the following command:

```
? MAX(123,4, 123,5)
```

dBASE Plus returns an error. The correct syntax is:

```
? MAX(123.4, 123.5)
```

SET POINT TO without the <expC> option resets the decimal character to the default set with theRegional settings of the Windows Control Panel.

SET PRECISION

Topic group Related topics Example

Determines the number of digits used when comparing numbers.

Syntax

SET PRECISION TO [<expN>]

<expN>

The number of digits, from 10 to 34. The default is 10.

Description

Use SET PRECISION to change the accuracy, or precision, of numeric comparisons. You can set precision from 10 to 34 digits.

SET PRECISION affects data comparisons, but not mathematical computations or data display. Math computations always use full precision internally. To change the number of decimal places *dBASE Plus* displays, use SET DECIMALS.

In general, you should use as little precision as possible for comparisons. Like many programs, *dBASE Plus* handles numbers as base-2 floating point numbers. This format precisely represents fractional values such as 0.5 (1/2) or 0.375 (3/8).

dBASE PLUS 9not supports High Precision Math and can handle large number correctly. To test the precision of the new dBASE implementaion here are a few tests from: http://www.sp4gl.com/index_perfect.htmlx

Examples: (in the COMMAND Window)

```
? 301.840002 - 301.000001
? 123456789.012345 * 87654321.123456789
```

OUTPUT:

```
0.840001
10821521028958940.344459595060205
```

SET SEPARATOR

Topic group Related topics

Specifies the character that separates each group of three digits (whole numbers) to the left of the decimal point in the display of numbers greater than or equal to 1000.

Syntax

SET SEPARATOR TO [<expC>]

<expC>

The whole-number separator, which is the character that separates each group of three digits to the left of the decimal point in the display of numbers greater than or equal to 1000. You can specify more than one character, but *dBASE Plus* uses only the first one. If you specify a number as a character for <expC> (for example, "3"), *dBASE Plus* returns an error.

The default is set by the Regional Settings of the Windows Control Panel.

Description

SET SEPARATOR affects only the PICTURE "," template character and the numeric display of byte totals for the commands such as DIR, DISPLAY FILES, and LIST FILES. For example, if you SET SEPARATOR TO "."(period) and issue the following, *dBASE Plus* returns 123456 displayed as 123.456:

```
? 123456 PICTURE "999,999"
```

You must use the comma in the PICTURE function, regardless of the setting of SET SEPARATOR.

SET SEPARATOR TO without the <expC> option resets the separator to the default set with theRegional Settings of the Windows Control Panel.

Setting a whole-number separator with SET SEPARATOR doesn't affect the values of numbers, only their display.

SIGN( )

Topic group Related topics Example

Returns an integer that indicates if a specified number is positive, negative, or zero (0).

Syntax

SIGN(<expN>)

<expN>

The number whose sign (positive, negative, or zero) to determine.

Description

Use SIGN( ) to reduce an arbitrary numeric value into one three numbers: 1, -1, or zero. SIGN( ) returns 1 if a specified number is positive, -1 if that number is negative, and 0 if that number is 0.

SIGN( ) is used when the numbers 1, -1, and/or 0 are appropriate for an action, based on the sign—but not the magnitude—of another number. When interested in the sign alone, it’s more straightforward to compare the number with zero using a comparison operator.

SIGN( ) always returns an integer, regardless of the value of SET DECIMALS.

SIN( )

Topic group Related topics

Returns the trigonometric sine of an angle.

Syntax

SIN(<expN>)

<expN>

The size of the angle in radians. To convert an angle’s degree value to radians, use DTOR( ). For example, to find the sine of a 30-degree angle, use SIN(DTOR(30)).

Description

SIN( ) calculates the ratio between the side opposite an angle and the hypotenuse in a right triangle. SIN( ) returns a number from –1 to +1. SIN( ) returns zero when <expN> is zero, pi, or 2pi radians.

Use SET DECIMALS to set the number of decimal places SIN( ) displays.

The cosecant of an angle is the reciprocal of the sine of the angle. To return the cosecant of an angle, use 1/SIN( ).

SQRT( )

Topic group Related topics

Returns the square root of a number.

Syntax

SQRT(<expN>)

<expN>

A positive number whose square root you want to return. If <expN> is a negative number, *dBASE Plus* generates an error.

Description

SQRT( ) returns the positive square root of a non-negative number. For example SQRT(36) returns 6 because 6^2 = 36. The square root of 0 is 0.

An alternate way to find the square root is to raise the value to the power of 0.5. For example, the following two statements display the same value:

```
? sqrt(36) ) // displays 6.00
? 36^.5 // displays 6.00
```

Use SET DECIMALS to set the number of decimal places SQRT( ) displays.

TAN( )

Topic group Related topics

Returns the trigonometric tangent of an angle.

Syntax

TAN(<expN>)

<expN>

The size of the angle in radians. To convert an angle's degree value to radians, use DTOR( ). For example, to find the tangent of a 30-degree angle, use TAN(DTOR(30)).

Description

TAN( ) calculates the ratio between the side opposite an angle and the side adjacent to the angle in a right triangle. TAN( ) returns a number that increases from zero to plus or minus infinity. TAN( ) returns zero when <expN> is 0, pi, or 2*pi radians. TAN( ) is undefined (returns infinity) when <expN> is pi/2 or 3*pi/2 radians.

Use SET DECIMALS to set the number of decimal places TAN( ) displays.

The cotangent of an angle is the reciprocal of the tangent of the angle. To return the cotangent of an angle, use 1/TAN( ).

Strings

String object

Topic group

dBASE Plus supports two types of strings:

- A primitive string that is compatible with earlier versions of dBASE
- A JavaScript-compatible String object.

dBASE Plus will convert one type of string to the other on-the-fly as needed. For example, you may use a String class method on a primitive string value or a literal string:

```
? version( ).toUpperCase( )
? "peter piper pickles".toProperCase( )
```

This creates a temporary String object from which the method or property is called. You may also use a string function on a String object.

Many string object methods have built-in string functions that are practically identical, while others are merely similar with subtle differences.

Note

JavaScript is zero-based; dBL is one-based. For example, to extract a substring starting with the third character, you would use the parameter 2 with the *substring*( ) method, and the parameter 3 with the SUBSTR( ) function.

The maximum length of a string in dBL is approximately 1 billion characters, or the amount of virtual memory, whichever is less.

class String

Topic group Example

A string of characters.

Syntax

[<oRef> =] new String([<expC>])

<oRef>

A variable or property in which you want to store a reference to the newly created String object.

<expC>

The string you want to create. If omitted or *null*, the resulting string is empty.

Properties

The following tables list the properties and methods of the String class. (No events are associated with this class.)

Property	Default	Description
baseClassName	STRING	Identifies the object as an instance of the String class
className	STRING	Identifies the object as an instance of the String class
length		The number of characters in the string
string		The value of the String object

Method	Parameters	Description
anchor( )	<expC>	Tags the string as an anchor <A NAME>
asc( )	<expC>	Returns the ASCII value of the first character in the designated string
big( )		Tags the string as big <BIG>
blink( )		Tags the string as blinking <BLINK>
bold( )		Tags the string as bold <BOLD>
charAt( )	<index expN>	Returns the character in the string at the designated position
chr( )	<expN>	Returns the character equivalent of the specified ASCII value

fixed( )		Tags the string as fixed font <TT>
fontcolor( )	<expC>	Tags the string as the designated color
fontsize( )	<expN>	Tags the string as the designated font size
getByte( )	<index expN>	Returns the value of the byte at the specified index in the string
indexOf( )	<expC> [, <start index expN>]	Returns the position of the search string inside the string
isAlpha( )		Returns *true* if the first character of the string is alphabetic
isLower( )		Returns *true* if the first character of the string is lowercase
isUpper( )		Returns *true* if the first character of the string is uppercase
italics( )		Tags the string as in italics <I>
lastIndexOf( )	<expC> [, <start index expN>]	Returns the position of the search string inside the string, searching backwards
left( )	<expN>	Returns the specified number of characters from the beginning of the string
leftTrim( )		Returns the string with all leading spaces removed
link( )	<expC>	Tags the string as a link <A HREF>
replicate( )	<expC> [, <expN>]	Returns the specified string repeated a number of times
right( )	<expN>	Returns the specified number of characters from the end of the string
rightTrim( )		Returns the string with all trailing spaces removed
setByte( )	<*index* expN>, <*value* expN>	Assigns a new value to the byte at the specified index in the string
small( )		Tags the string as small <SMALL>
space( )	<expN>	Returns a string comprising the specified number of spaces
strike( )		Tags the string as strikethrough <STRIKE>
stuff( )	<start expN> , <quantity expN> [, <replacement expC>]	Returns the string with specified characters removed and others inserted in their place
sub( )		Tags the string as subscript <SUB>
substring( )	<start index expN> , <end index expN>	Returns a substring derived from the string
sup( )		Tags the string as superscript <SUP>
toLowerCase( )		Returns the string in all lowercase
toProperCase( )		Returns the string in proper case
toUpperCase( )		Returns the string in all uppercase

Description

A String object contains the actual string value, stored in the property *string*, and methods that act upon that value. The methods do not modify the value of *string*; they use it as a base and return another string, number, or *true* or *false*.

The methods are divided into three categories: those that simply wrap the string in HTML tags, those that act upon the contents of the string, and static class methods that do not operate on the string at all.

Because the return values for most string methods are also strings, you can call more than one method for a particular string by chaining the method calls together. For example,

```
cSomething.substring( 4, 7 ).toUpperCase( )
```

ASC( )

Topic group Related topics Example

Returns the numeric ASCII value of a specified character.

Syntax

ASC(<expC>)

<expC>

The character whose ASCII value you want to return. You can specify a string with more than one character, but *dBASE Plus* uses only the first one.

Description

ASC( ) is the inverse of CHR( ). ASC( ) accepts a character and returns its ASCII value—a number from 0 to 255, inclusive. CHR( ) accepts an ASCII value and returns its character.

Other than the syntactic difference of being a method instead of a function, the *asc*( ) method behaves identically to the ASC( ) function.

asc( )

Topic group Related topics

Returns the numeric ASCII value of a specified character.

Syntax

<oRef>.*asc*(<expC>)

<oRef>

A reference to a String object.

<expC>

The character whose ASCII value you want to return. You can specify a string with more than one character, but *dBASE Plus* uses only the first one.

Property of

String

Description

Other than the syntactic difference of being a method instead of a function, this method behaves identically to the ASC( ) function.

AT( )

Topic group Related topics Example

Returns a number that represents the position of a string within another string.

Syntax

AT(<search expC>, <target expC> [, <nth occurrence expN>])

<search expC>

The string you want to search for in <target expC>.

<target expC>

The string in which to search for <search expC>.

<nth occurrence expN>

Which occurrence of the string to find. By default, *dBASE Plus* searches for the first occurrence. You can search for other occurrences by specifying the number, which must be greater than zero.

Description

AT( ) returns the numeric position where a search string begins in a target string. AT( ) searches one character at a time from left to right, beginning to end, from the first character to the last character. The search is case-sensitive. Use UPPER( ) or LOWER( ) to make the search case-insensitive.

You can specify which occurrence of the search string to find by specifying a number for <nth occurrence expN>. If you omit this argument, AT( ) returns the starting position of the first occurrence of the search string.

AT( ) returns 0 when

- The search string isn't found.
- The search string or target string is empty.
- The search string is longer than the target string.
- The <nth occurrence expN> occurrence doesn't exist.

When AT( ) counts characters in a memo field, it counts two characters for each carriage-return and linefeed combination (CR/LF) in the memo field.

Use RAT( ) to find the starting position of <search expC>, searching from right to left, end to beginning. Use the substring operator ($) to learn if one string exists within another.

The *indexOf( )* method is similar to the AT( ) function, but in addition to the syntactic difference of being a method instead of a function and the fact that the position is zero-based, the method's optional parameter specifies where to start searching instead of the nth occurrence to find.

CENTER( )

Topic group Related topics

Returns a character string that contains a string centered in a line of specified length.

Syntax

CENTER(<expC> [, <length expN> [, <pad expC>]])

<expC>

The text to center.

<length expN>

The length of the resulting line of text. The default is 80 characters.

<pad expC>

The single character to pad the string with if <length expN> is greater than the number of characters in <expC>. If <length expN> is equal to or less than the number of characters in <expC>, <pad expC> is ignored.

If <pad expC> is more than one character, CENTER( ) uses only the first character. If <pad expC> isn't specified, CENTER( ) pads with spaces.

Description

CENTER( ) returns a character expression with the requisite number of leading and trailing spaces to center it in a line that is a specified number of characters wide.

To create the resulting string, CENTER( ) performs the following steps.

Subtracts the length of <expC> or <memo field> from <length expN>
Divides the result in half and rounds up if necessary
Pads <expC> on either side with that number of spaces or the first character in <pad expC>

If the length of <expC> or <memo field> is greater than <length expN>, CENTER( ) does the following:

Subtracts <length expN> from the length of <expC>
Divides the result in half and rounds up if necessary
Truncates both sides of <expC> by that many characters

When the result of subtracting the length of <expC> from <length expN> is an odd number, CENTER( ) pads one less space on the left if the difference is positive, or truncates one less character on the left if the difference is negative.

charAt( )

Topic group Related topics

Returns the character at the specified position in the string.

Syntax

<expC>.*charAt*(<expN>)

<expC>

A string.

<expN>

Index into the string, which is indexed from left to right. The first character of the string is at index 0 and the last character is at index <expC>.length – 1.

Property of

String

Description

charAt( ) returns the character in a String object at the specified index position. If the index position is after the last character in the string, *charAt*( ) returns an empty string.

CHR( )

Topic group Related topics Example

Returns the character equivalent of a specified ASCII value.

Syntax

CHR(<expN>)

<expN>

The numeric ASCII value, from 0 to 255, inclusive, whose character equivalent you want to return.

Description

CHR( ) is the inverse of ASC( ). CHR( ) accepts an ASCII value and returns its character, while ASC( ) accepts a character and returns its ASCII value.

Other than the syntactic difference of being a method instead of a function, the *chr( )* method behaves identically to the CHR( ) function.

chr( )

Topic group Related topics

Returns the character equivalent of a specified ASCII value.

Syntax

<oRef>.*chr*(<expN>)

<oRef>

A reference to a String object.

<expN>

The numeric ASCII value, from 0 to 255, inclusive, whose character equivalent you want to return.

Property of

String

Description

Other than the syntactic difference of being a method instead of a function, this method behaves identically to the CHR( ) function.

DIFFERENCE( )

Topic group Related topics Example

Returns a number that represents the phonetic difference between two strings.

Syntax

DIFFERENCE(<expC1>, <expC2>)

<expC1>

The first character expression to evaluate the SOUNDEX( ) of and compare to the second value.

<expC2>

The second character expression to evaluate the SOUNDEX( ) of and compare to the first value.

Description

SOUNDEX( ) returns a four-character code that represents the phonetic value of a character expression. DIFFERENCE( ) compares the SOUNDEX( ) codes of two character expressions, and returns an integer from 0 to 4 that expresses the difference between the codes.

A returned value of 0 indicates the greatest difference in SOUNDEX( ) codes—the two expressions have no SOUNDEX( ) characters in common. A returned value of 4 indicates the least difference—the two expressions have all four SOUNDEX( ) characters in common. However, using DIFFERENCE( ) on short strings can produce unexpected results, as shown in the following example.

```
? soundex( "Mom" ) // Displays M500
? soundex( "Dad" ) // Displays D300
? difference( "Mom", "Dad" ) // Displays 2
```

To compare the character-by-character similarity between two strings rather than the phonetic similarity, use LIKE( ).

getByte( )

Topic group Related topics Example

Returns the value of the byte at the specified index in the string.

Syntax

<oRef>.*getByte*(<index expN>)

<oRef>

A reference to the String object that you're using as a structure.

<index expN>

The index number of the desired byte. The first byte is at index number zero.

Property of

String

Description

Strings in dBL are Unicode strings, which use double-byte characters. Use *getByte*( ) when using a string as a structure that is passed to a DLL function that you have prototyped with EXTERN, to get the values of the bytes in the structure.

indexOf( )

Topic group Related topics

Returns a number that represents the position of a string within another string.

Syntax

<target expC>.*indexOf*(<search expC> [, <from index expN>])

<target expC>

The string in which you want to search for <search expC>.

<search expC>

The string you want to search for in <target expC>.

<from index expN>

Where you want to start searching for the string. By default, *dBASE Plus* starts searching at the beginning of the string, index 0.

Property of

String

Description

This method is similar to the AT( ) function, but in addition to the syntactic difference of being a method instead of a function and the fact that the position is zero-based, the optional parameter specifies where to start searching instead of the nth occurrence to find.

indexOf( ) returns an index representing where a search string begins in a target string. The first character of the string is at index 0 and the last character is at index <target expC>.length – 1. *indexOf*( ) searches one character at a time from left to right, beginning to end, from the character at <from index expN> to the last character. The search is case-sensitive. Use *toUpperCase*( ) or *toLowerCase*( ) to make the search case-insensitive.

indexOf( ) returns –1 when

- The search string isn't found.
- The search string or target string is empty.
- The search string is longer than the target string.

Use *lastIndexOf*( ) to find the starting position of <search expC>, searching from right to left, end to beginning.

ISALPHA( )

Topic group Related topics

Returns *true* if the first character of a string is alphabetic.

Syntax

ISALPHA(<expC>)

<expC>

The string you want to test.

Description

ISALPHA( ) tests the first character of the string and returns *true* if it's an alphabetic character. ISALPHA( ) returns *false* if the character isn't alphabetic or if that character position is empty.

The current language driver defines the character values that are lowercase and uppercase alphabetic. In a U.S. language driver, a lowercase alphabetic character is from a to z, and an uppercase alphabetic character is from A to Z.

Other than the syntactic difference of being a method instead of a function, the *isAlpha*( ) method behaves identically to the ISAPLHA( ) function.

isAlpha( )

Topic group Related topics

Returns *true* if the first character of a string is alphabetic.

Syntax

<expC>.*isAlpha*( )

<expC>

The string you want to test.

Property of

String

Description

Other than the syntactic difference of being a method instead of a function, this method behaves identically to the ISAPLHA( ) function.

ISLOWER( )

Topic group Related topics

Returns *true* if the first character of a string is alphabetic and lowercase.

Syntax

ISLOWER(<expC>)

<expC>

The string you want to test.

Description

ISLOWER( ) tests the first character of the string and returns *true* if it's a lowercase alphabetic character. ISLOWER( ) returns *false* if the character isn't lowercase or if the character expression is empty.

The current language driver defines the character values that are lowercase and uppercase alphabetic. In a U.S. language driver, a lowercase alphabetic character is from a to z, and an uppercase alphabetic character is from A to Z.

Other than the syntactic difference of being a method instead of a function, the *isLower*( ) method behaves identically to the ISLOWER( ) function.

isLower( )

Topic group Related topics

Returns *true* if the first character of a string is alphabetic and lowercase.

Syntax

<oRef>.*isLower*( )

<expC>

The string you want to test.

Property of

String

Description

Other than the syntactic difference of being a method instead of a function, this method behaves identically to the ISLOWER( ) function.

ISUPPER( )

Topic group Related topics

Returns *true* if the first character of a string is alphabetic and uppercase.

Syntax

ISUPPER(<expC>)

<expC>

The string you want to test.

Description

ISUPPER( ) tests the first character of the string and returns *true* if it's an uppercase alphabetic character. ISUPPER( ) returns *false* if the character isn't uppercase or if the character expression is empty.

The current language driver defines the character values that are lowercase and uppercase alphabetic. In a U.S. language driver, a lowercase alphabetic character is from a to z, and an uppercase alphabetic character is from A to Z.

Other than the syntactic difference of being a method instead of a function, the *isUpper*( ) method behaves identically to the ISUPER( ) function.

isUpper( )

Topic group Related topics

Returns *true* if the first character of a string is alphabetic and uppercase.

Syntax

<expC>.*isUpper*( )

expCThe string you want to test.

Property of

String

Description

Other than the syntactic difference of being a method instead of a function, this method behaves identically to the ISUPPER( ) function.

lastIndexOf( )

Topic group Related topics

Returns a number that represents the starting position of a string within another string. *lastIndexOf*( ) searches backward from the right end of the target string, and returns a value counting from the beginning of the target.

Syntax

<target expC>.*lastIndexOf*(<search expC> [, <from index expN>])

<target expC>

The string in which you want to search for <search expC>.

<search expC>

The string you want to search for in <target expC>.

<from index expN>

Where you want to start searching for the string. By default, *dBASE Plus* starts searching at the end of the string, index <target expC>.length – 1.

Property of

String

Description

This method is similar to the RAT( ) function, but in addition to the syntactic difference of being a method instead of a function and the fact that the position is zero-based, the optional parameter specifies where to start searching instead of the nth occurrence to find.

Use *lastIndexOf*( ) to search for the <search expC> in a target string, searching right to left, end to beginning, from the character at <from index expN> to the first character. The search is case-sensitive. Use *toUpperCase( )* or *toLowerCase( )* to make the search case-insensitive.

Even though the search starts from the end of the target string, *lastIndexOf*( ) returns an index representing where a search string begins in a target string, counting from the beginning of the target. The first character of the string is at index 0 and the last character is at index <target expC>.length – 1. If <search expC> occurs only once in the target, *lastIndexOf*( ) and *indexOf*( ) return the same value. For example, "abcdef".*lastIndexOf*("abc") and "abcdef".*indexOf*("abc") both return 0.

lastIndexOf( ) returns –1 when:

- The search string isn't found
- The search string or target string is empty
- The search string is longer than the target string

To find the starting position of <search expC>, searching from left to right, beginning to end, use *indexOf( )*.

LEFT( )

Topic group Related topics Example

Returns a specified number of characters from the beginning of a string.

Syntax

LEFT(<expC>, <expN>)

<expC>

The string from which you want to extract characters.

<expN>

The number of characters to extract from the beginning of the string.

Description

Starting with the first character of a character expression, LEFT( ) returns <expN> characters. If <expN> is greater than the number of characters in the specified string, LEFT( ) returns the string as it is, without adding spaces to achieve the specified length. You can use LEN( ) to determine the actual length of the returned string.

If <expN> is less than or equal to zero, LEFT( ) returns an empty string. If <expN> is greater than or equal to zero, LEFT(<expC>, <expN>) achieves the same results as SUBSTR(<expC>, 1, <expN>).

When LEFT( ) returns characters from a memo field, it counts two characters for each carriage-return and linefeed combination (CR/LF).

Other than the syntactic difference of being a method instead of a function, the *left*( ) method behaves identically to the LEFT( ) function.

left( )

Topic group Related topics

Returns a specified number of characters from the beginning of a character string.

Syntax

<expC>.*left*(<expN>)

<expC>

The string from which you want to extract characters.

<expN>

The number of characters to extract from the beginning of the string.

Property of

String

Description

Other than the syntactic difference of being a method instead of a function, this method behaves identically to the LEFT( ) function.

leftTrim( )

Topic group Related topics

Returns a string with no leading space characters.

Syntax

<expC>.*leftTrim*( )

<expC>

The string from which you want to remove the leading space characters.

Property of

String

Description

Other than the syntactic difference of being a method instead of a function, this method behaves identically to the LTRIM( ) function.

LEN( )

Topic group Related topics

Returns the number of characters in a specified character string.

Syntax

LEN(<expC>)

<expC>

The character string whose length you want to find.

Description

LEN( ) returns the number of characters (the length) of a character string or memo field. The length of an empty character string or empty memo field is zero. When LEN( ) calculates the length of a memo field, it counts two characters for each carriage-return and linefeed combination (CR/LF).

Other than the syntactic difference of reading a property instead of calling a function, *length* contains the same value that LEN( ) returns.

length

Topic group Related topics

The number of characters in a specified character string.

Syntax

<expC>.*length*

<expC>

The character string whose length you want to find.

Property of

String

Description

A string's *length* property reflects the number of characters (the length) of a character string. The length of an empty character string is zero.

length is a read-only property.

Other than the syntactic difference of reading a property instead of calling a function, *length* contains the same value that LEN( ) returns.

LENNUM( )

Topic group Related topics

Returns the display length (in characters) of a numeric expression.

Syntax

LENNUM(<expN>)

<expN>

The numeric or float number whose display length to return.

Description

Use LENNUM( ) before formating a display involving numeric values of varying lengths.

If you pass LENNUM( ) the name of a numeric field, it returns the length of the field.

If a number has eight or fewer whole-number digits and no decimal point, it is by default a numeric-type number; the default display length for numeric-type numbers is 11. For example:

```
?LENNUM(123)
```

returns 11.

If a number contains a decimal point, the value returned by LENNUM( ) will depend on the value of SET DECIMALS TO. The minimum value returned will be comprised of the minimum default length (11), a character for the decimal point (1) plus the value of SET DECIMALS TO (regardless of how many decimal places are actually utilized). Therefore, where SET DECIMALS TO = 2;

```
?LENNUM(122.1)
```

and

```
?LENNUM(122.11)
```

both return 14.

The maximum length returned by LENNUM( ) is 39.

If a number passed to LENNUM( ) is null, LENNUM( ) returns a null “value”.

LIKE( )

Topic group Related topics

Returns *true* if a specified string matches a specified skeleton string.

Syntax

LIKE(<skeleton expC>, <expC>)

<skeleton expC>

A string containing a combination of characters and wildcards. The wildcards are ? and *.

<expC>

The string to compare to the skeleton string.

Description

Use LIKE( ) to compare one string to another. The <skeleton expC> argument contains wildcard characters and represents a pattern; the <expC> argument is compared to this pattern. LIKE( ) returns *true* if <expC> is a string that matches <skeleton expC>. To compare the phonetic similarity between two strings rather than the character-by-character similarity, use DIFFERENCE( ).

Use the wildcard characters ? and * to form the pattern for <skeleton expC>. An asterisk (*) stands for any number of characters, including zero characters. The question mark (?) stands for any single character. Both wildcards can appear anywhere and more than once in <skeleton expC>. Wildcard characters in <skeleton expC> can stand for uppercase or lowercase letters.

If * or ? appears in <expC>, they are interpreted as literal, not wildcard, characters, as shown in the following example.

```
? LIKE( "a*d", "abcd" ) // Displays true
? LIKE( "a*d", "aBCd" ) // Displays true
? LIKE( "abcd", "a*d" ) // Displays false
```

LIKE( ) is case-sensitive. Use UPPER( ) or LOWER( ) for case-insensitive comparisons with LIKE( ). LIKE( ) returns *true* if both arguments are empty strings. LIKE( ) returns *false* if one argument is empty and the other isn't.

LOWER( )

Topic group Related topics

Converts all uppercase characters in a string to lowercase and returns the resulting string.

Syntax

LOWER(<expC>)

<expC>

The string you want to convert to lowercase.

Description

LOWER( ) converts the uppercase alphabetic characters in a character expression or memo field to lowercase. LOWER( ) ignores digits and other characters.

The current language driver defines the character values that are lowercase and uppercase alphabetic. In a U.S. language driver, a lowercase alphabetic character is from a to z, and an uppercase alphabetic character is from A to Z.

Other than the syntactic difference of being a method instead of a function, the *toLowerCase( )* method behaves identically to the LOWER( ) function.

LTRIM( )

Topic group Related topics

Returns a string with no leading space characters.

Syntax

LTRIM(<expC>)

<expC>

The string from which you want to remove the leading space characters.

Description

LTRIM( ) returns <expC> with no leading space characters.

To remove trailing space characters from a string or memo field, use RTRIM( ) or TRIM( ).

Other than the syntactic difference of being a method instead of a function, the *leftTrim( )* method behaves identically to the LTRIM( ) function.

PROPER( )

Topic group Related topics

Converts a character string to proper-noun format and returns the resulting string.

Syntax

PROPER(<expC>)

<expC>

The character string to convert to proper-noun format.

Description

PROPER( ) returns <expC> with the first character in each word capitalized and the remaining letters set to lowercase. PROPER( ) changes the first character of a word only if it is a lowercase alphabetic character.

The current language driver defines the character values that are lowercase and uppercase alphabetic. In a U.S. language driver, a lowercase alphabetic character is from a to z, and an uppercase alphabetic character is from A to Z.

Other than the syntactic difference of being a method instead of a function, the *toProperCase( )* method behaves identically to the PROPER( ) function.

RAT( )

Topic group Related topics Example

Returns a number that represents the starting position of a string within another string. RAT( ) searches backward from the right end of the target string, and returns a value counting from the beginning of the target.

Syntax

RAT(<search expC>, <target expC> [, <nth occurrence expN>])

<search expC>

The string you want to search for in <target expC>.

<target expC>

The string in which to search for <search expC>.

<nth occurrence expN>

Which occurrence of the string to find. By default, *dBASE Plus* searches for the first occurrence from the end. You can search for other occurrences by specifying the number (based on starting from the end), which must be greater than zero.

Description

Use RAT( ) to search for the first or <nth occurrence expN> occurrence of <search expC> in a target string, searching right to left, end to beginning, from the last character to the first character. The search is case-sensitive. Use UPPER( ) or LOWER( ) to make the search case-insensitive.

Even though the search starts from the end of the target string or memo field, RAT( ) returns the numeric position where a search string begins in a target string, counting from the beginning of the target. If <search expC> occurs only once in the target, RAT( ) and AT( ) return the same value. For example, RAT("abc","abcdef") and AT("abc","abcdef") both return 1.

RAT( ) returns 0 when:

- The search string isn't found
- The search string or target string is empty
- The search string is longer than the target string
- The nth occurrence you specify with <nth occurrence expN> doesn't exist

To find the starting position of <search expC>, searching from left to right, beginning to end, use AT( ). To learn if one string exists within another, use the substring operator ($).

The *lastIndexOf*( ) method is similar to the RAT( ) function, but in addition to the syntactic difference of being a method instead of a function and the fact that the position is zero-based, the optional parameter specifies where to start searching instead of the nth occurrence to find.

REPLICATE( )

Topic group Related topics Example

Returns a string repeated a specified number of times.

Syntax

REPLICATE(<expC>, <expN>)

<expC>

The string you want to repeat.

<expN>

The number of times to repeat the string.

Description

REPLICATE( ) returns a character string composed of a character expression repeated a specified number of times.

If the character expression is an empty string, REPLICATE( ) returns an empty string. If the number of repeats you specify for <expN> is 0 or less, REPLICATE( ) returns an empty string.

To repeat space characters, use SPACE( ).

The *replicate*( ) method is almost identical to the REPLICATE( ) function, but in addition to the syntactic difference of being a method instead of a function, the repeat count is optional and defaults to 1.

replicate( )

Topic group Related topics

Returns a string repeated a specified number of times.

Syntax

<oRef>.*replicate*(<expC> [, <expN>])

<oRef>

A reference to a String object.

<expC>

The string you want to repeat.

<expN>

The number of times to repeat the string; by default, 1.

Property of

String

Description

This method is almost identical to the REPLICATE( ) function, but in addition to the syntactic difference of being a method instead of a function, the repeat count is optional and defaults to 1.

RIGHT( )

Topic group Related topics

Returns characters from the end of a character string.

Syntax

RIGHT(<expC>, <expN>)

<expC>

The string from which you want to extract characters.

<expN>

The number of characters to extract from the string.

Description

Starting with the last character of a character expression, RIGHT( ) returns a specified number of characters. If the number of characters you specify for <expN> is greater than the number of characters in the specified string or memo field, RIGHT( ) returns the string as is, without adding spaces to achieve the specified length. If <expN> is less than or equal to zero, RIGHT( ) returns an empty string.

Strings often have trailing blanks. You may want to remove them with TRIM( ) before using RIGHT( ).

When RIGHT( ) returns characters from a memo field, it counts two characters for each carriage-return and linefeed combination (CR/LF).

Other than the syntactic difference of being a method instead of a function, the *right*( ) method behaves identically to the RIGHT( ) function.

right( )

Topic group Related topics

Returns characters from the end of a character string.

Syntax

<expC>.*right*(<expN>)

<expC>

The string from which you want to extract characters.

<expN>

The number of characters to extract from the string.

Property of

String

Description

Other than the syntactic difference of being a method instead of a function, this method behaves identically to the RIGHT( ) function.

rightTrim( )

Topic group Related topics

Returns a string with no trailing space characters.

Syntax

<expC>.*rightTrim*( )

<expC>

The string from which you want to remove the trailing space characters.

Property of

String

Description

Other than the syntactic difference of being a method instead of a function, this method behaves identically to the TRIM( ) function.

RTRIM( )

Topic group Related topics

Returns a string with no trailing space characters.

Syntax

RTRIM(<expC>)

<expC>

The string from which you want to remove the trailing space characters.

Description

RTRIM( ) is identical to TRIM( ). See TRIM( ) for details.

setByte( )

Topic group Related topics Example

Assigns a new value to the byte at the specified index in the string.

Syntax

<oRef>.*setByte*(<index expN>, <value expN>)

<oRef>

A reference to the String object that you're using as a structure.

<index expN>

The index number of the byte to set. The first byte is at index number zero.

<value expN>

The new byte value, from 0 to 255.

Property of

String

Description

Strings in dBL are Unicode strings, which use double-byte characters. Use *setByte*( ) when using a string as a structure that is passed to a DLL function that you have prototyped with EXTERN, to set the values of the bytes in the structure.

The *length* of the structure string should be one-half the number of bytes in the structure, rounded up. Setting the individual bytes of a Unicode string will most likely cause the string to become unprintable.

SOUNDEX( )

Topic group Related topics Example

Returns a four-character string that represents the SOUNDEX (sound-alike) code of another string.

Syntax

SOUNDEX(<expC>)

<expC>

The string for which to calculate the soundex code.

Description

SOUNDEX( ) returns a four-character code that represents the phonetic value of a character expression. The code is in the form "letter digit digit digit," where "letter" is the first alphabetic character in the expression being evaluated. The more phonetically similar two strings are, the more similar their SOUNDEX codes.

Use SOUNDEX( ) to find words that sound similar, or are spelled similarly, such as names like "Smith," "Smyth," and "Smythe." Using the U.S. language driver, these all evaluate to S531.

SOUNDEX( ) returns "0000" if the character expression is an empty string or if the first nonblank character isn't a letter. SOUNDEX( ) returns 0's for the first digit encountered and for all following characters, regardless of whether they're digits or alphabetic characters.

To compare the SOUNDEX values of two character expressions or memo fields, use DIFFERENCE( ). If you want to compare the character-by-character similarity between two strings rather than the phonetic similarity, use LIKE( ).

SOUNDEX( ) is language driver-specific. If the current language driver is U.S., SOUNDEX( ) does the following to calculate the phonetic value of a string:

- Ignores leading spaces.
- Ignores the letters A, E, I, O, U, Y, H, and W.
- Ignores case.
- Converts the first nonblank character to uppercase and makes it the first character in the SOUNDEX code.
- Converts B, F, P, and V to 1.
- Converts C, G, J, K, Q, S, X, and Z to 2.
- Converts D and T to 3.
- Converts L to 4.
- Converts M and N to 5.
- Converts R to 6.
- Removes the second occurrence of any adjacent letters that receive the same digits as phonetic values.
- Pads the end of the resulting string with zeros if fewer than three digits remain.
- Truncates the resulting string to three digits if more than three digits remain.
- Concatenates the first character of the code to the remaining three digits to create the "letter digit digit digit" soundex code.

SPACE( )

Topic group Related topics

Returns a specified number of space characters.

Syntax

SPACE(<expN>)

<expN>

The number of spaces you want to return.

Description

SPACE( ) returns a character string composed of a specified number of space characters. The space character is ASCII code 32.

If <expN> is 0 or less, SPACE( ) returns an empty string.

To create a string using a character other than the space character, use REPLICATE( ).

Other than the syntactic difference of being a method instead of a function, the *space*( ) method behaves identically to the SPACE( ) function.

space( )

Topic group Related topics

Returns a specified number of space characters.

Syntax

<*oRef*>.*space*(<expN>)

<oRef>

A reference to a String object.

<expN>

The number of spaces you want to return.

Property of

String

Description

Other than the syntactic difference of being a method instead of a function, this method behaves identically to the SPACE( ) function.

STR( )

Topic group Related topics

Returns the character string equivalent of a specified numeric expression.

Syntax

STR(<expN> [, <length expN> [, <decimals expN> [, <expC>]]])

<expN>

The numeric expression to return as a character string.

<length expN>

The length of the character string to return. The valid range is 1 to 20, inclusive, and includes a decimal point, decimal digits, and minus sign characters. The default is 10. If <length expN> is smaller than the number of integer digits in <expN>, STR( ) returns asterisks (*).

<decimals expN>

The number of characters to reserve for decimal digits. The default and lowest allowable value is 0. If you do not specify a value for <decimals expN>, STR( ) rounds <expN> to the nearest whole number. If you want to specify a value for <decimals expN>, you must also specify a value for <length expN>.

<expC>

The character to pad the beginning of the returned character string with when the length of the returned string is less than <length expN> digits long. The default pad character is a space. If you want to specify a value for <expC>, you must also specify values for <length expN> and <decimals expN>. You can specify more than one character for <expC>, but STR( ) uses only the first one.

Description

Use STR( ) to convert a number to a string, so you can manipulate it as characters. For example, you can index on a numeric field in combination with a character field by converting the numeric field to character with STR( ).

dBASE Plus rounds and pads numbers to fit within parameters you set with <length expN> and <decimals expN>, following these rules:

- If <decimals expN> is smaller than the number of decimals in <expN>, STR( ) rounds to the most accurate number that will fit in <length expN>. For example, STR(10.765,5,1) returns " 10.8" (with a single leading space), and STR(10.765,5,2) returns "10.77".
- If <length expN> isn't large enough for <decimals expN> number of decimal places, STR( ) rounds <expN> to the most accurate number that will fit in <length expN>. For example, STR(10.765,4,3) returns "10.8".
- If <decimals expN> is larger than the number of decimals in <expN>, and <length expN> is larger than the returned string, STR( ) adds zeros (0) to the end of the returned string. *dBASE Plus* only adds enough zero to bring the number of decimal digits to a maximum of <decimals expN>.
- If the returned string is still shorter than <length expN>, dBASE pads the left to fill to the length of <length expN>. For example, STR(10.765,8,6) returns "10.76500" for a returned length of 8; STR(10.765,7,6) returns "10.7650" for a returned length of 7; and STR(10.765,12,6) returns " 10.765000" (with three leading spaces) for a returned length of 12.

To remove the leading spaces created by STR( ), use LTRIM( ). If you concatenate a number to a string with the + or - operators, *dBASE Plus* automatically converts the number to a string, using the number of decimal places specified by SET DECIMALS, and removes the leading spaces.

STUFF( )

Topic group Related topics

Returns a string with specified characters removed and others inserted in their place.

Syntax

STUFF(<target expC>, <start expN>, <quantity expN>, <replacement expC>)

<target expC>

The string you want to remove characters from and replace with new characters.

<start expN>

The character position in the string at which you want to start removing characters.

<quantity expN>

The number of characters you want to remove from the string.

<replacement expC>

The characters you want to insert in the string.

Description

STUFF( ) returns a target character expression with a replacement character string inserted at a specified position. Starting at the position you specify, <start expN>, STUFF( ) removes a specified number, <quantity expN>, of characters from the original string.

If the target character expression is an empty string, STUFF( ) returns the replacement string.

If <start expN> is less than or equal to 0, STUFF( ) treats <start expN> as 1. If <quantity expN> is less than or equal to 0, STUFF( ) inserts the replacement string at position <start expN> without removing any characters from the target.

If <start expN> is greater than the length of the target, STUFF( ) doesn't remove any characters and appends the replacement string to the end of the target.

If the replacement string is empty, STUFF( ) removes the characters specified by <quantity expN> from the target, starting at <start expN>, without adding characters.

The *stuff*( ) method is almost identical to the STUFF( ) function, but in addition to the syntactic difference of being a method instead of a function and the fact that the position is zero-based, the replacement string is optional, and defaults to an empty string.

stuff( )

Topic group Related topics

Returns a string with specified characters removed and others inserted in their place.

Syntax

<expC>.*stuff*(<start expN>, <quantity expN> [, <replacement expC>])

<expC>

The string in which you want to remove and replace characters.

<start expN>

The character position in the string at which you want to start removing characters.

<quantity expN>

The number of characters you want to remove from the string.

<replacement expC>

The characters you want to insert in the string. By default, this is an empty string.

Property of

String

Description

This method is almost identical to the STUFF( ) function. However, the *stuff*( ) method is zero-based (the function is one-based), a replacement string is optional, and the method defaults to an empty string.

SUBSTR( )

Topic group Related topics Example

Returns a substring derived from a specified character string.

Syntax

SUBSTR(<expC>, <start expN> [, <length expN>])

<expC>

The string you want to extract characters from.

<start expN>

The character position in the string to start extracting characters.

<length expN>

The number of characters to extract from the string.

Description

Starting in a character expression at the position you specify for <start expN>, SUBSTR( ) returns the number of characters you specify for <length expN>. If <start expN> is greater than the length of <expC>, or <length expN> is zero or a negative number, SUBSTR( ) returns an empty string.

If you don't specify <length expN>, SUBSTR( ) returns all characters starting from position <start expN> to the end of the string. If <length expN> is greater than the number of characters from <start expN> to the end of the string, SUBSTR( ) returns only as many characters as are left in the string, without adding space characters to achieve the specified length. You can use LEN( ) to determine the actual length of the returned string.

When SUBSTR( ) returns characters from a memo field, it counts two characters for each carriage-return and linefeed combination (CR/LF) in the memo field.

The *substring*( ) method is similar to the SUBSTR( ) function, but in addition to the syntactic difference of being a method instead of a function and the fact that the position is zero-based, the method takes a starting and ending position, while the function takes a start position and the number of character to extract.

substring( )

Topic group Related topics

Returns a substring derived from a specified character string.

Syntax

<expC>.*substring*(<index1 expN>, <index2 expN>)

<expC>

The string you want to extract characters from.

<index1 expN>, <index2 expnN>

Indexes into the string, which is indexed from left to right. The first character of the string is at index 0 and the last character is at index <expC>.length – 1.

Property of

String

Description

This method is similar to the SUBSTR( ) function, but in addition to the syntactic difference of being a method instead of a function and the fact that the position is zero-based, the method takes a starting and ending position, while the function takes a start position and the number of character to extract.

<index1 expN> and <index2 expN> determine the position of the substring to extract. *substring*( ) begins at the lesser of the two indexes and extracts up to the character before the other index. If the two indexes are the same, *substring*( ) returns an empty string.

If the starting index is after the last character in the string, *substring*( ) returns an empty string.

toLowerCase( )

Topic group Related topics

Converts all uppercase characters in a string to lowercase and returns the resulting string.

Syntax

<expC>.*toLowerCase*( )

<expC>

The string you want to convert to lowercase.

Property of

String

Description

Other than the syntactic difference of being a method instead of a function, this method behaves identically to the LOWER( ) function.

toProperCase( )

Topic group Related topics

Converts a character string to proper-noun format and returns the resulting string.

Syntax

<expC>.*toProperCase*( )

<expC>

The string you want to convert to proper-noun format.

Property of

String

Description

Other than the syntactic difference of being a method instead of a function, this method behaves identically to the PROPER( ) function.

toUpperCase( )

Topic group Related topics

Converts all lowercase characters in a string to uppercase and returns the resulting string.

Syntax

<expC>.*toUpperCase*( )

<expC>

The string you want to convert to uppercase.

Property of

String

Description

Other than the syntactic difference of being a method instead of a function, this method behaves identically to the UPPER( ) function.

TRANSFORM( )

Topic group Related topics Example

Applies a formatting template to an expression, returning a formatted string.

Syntax

TRANSFORM(<exp>, <picture expC>)

<exp>

The expression to be formatted.

<picture expC>

The string containing the template characters necessary to format <exp>. The template characters are the same characters used in the *picture* property of an entryfield.

Description

TRANSFORM( ) returns an expression in the template format you indicate with <picture expC>.

TRIM( )

Topic group Related topics

Returns a string with no trailing space characters.

Syntax

TRIM(<expC>)

<expC>

The string from which you want to remove the trailing space characters.

Description

TRIM( ) returns a character expression with no trailing space characters. TRIM( ) is identical to RTRIM( ).

To remove trailing blanks before concatenating a string to another string, use the - operator instead of the + operator.

Warning!

Do not create index expressions with TRIM( ) that result in key values that vary in length from record to record. This results in unbalanced indexes that may become corrupted. Use the - operator, which relocates trailing blanks without changing the resulting length of the concatenated string.

To remove leading space characters from a string, use LTRIM( ).

Other than the syntactic difference of being a method instead of a function, the *rightTrim*( ) method behaves identically to the TRIM( ) function.

UPPER( )

Topic group Related topics Example

Converts all lowercase characters in a string to uppercase and returns the resulting string.

Syntax

UPPER(<expC>)

<expC>

The character string you want to convert to uppercase.

Description

UPPER( ) converts the lowercase alphabetic characters in a character expressionor memo field to uppercase. UPPER( ) ignores digits and other characters.

The current language driver defines the character values that are lowercase and uppercase alphabetic. In a U.S. language driver, a lowercase alphabetic character is from a to z, and an uppercase alphabetic character is from A to Z.

Other than the syntactic difference of being a method instead of a function, the *toUpperCase*( ) method behaves identically to the UPPER( ) function.

VAL( )

Topic group Related topics

Returns the number at the beginning of a character string.

Syntax

VAL(<expC>)

<expC>

The character expression that contains the number.

Description

Use VAL( ) to convert a string that contains a number into an actual number. Once you convert a string to a number, you can perform arithmetic operations with it.

If the character string you specify contains both letters and numbers, VAL( ) returns the value of the entire number to the left of the first nonnumeric character. If the string contains a nonnumeric character other than a blank space in the first position, VAL( ) returns 0. For example, VAL("ABC123ABC456") returns 0, VAL("123ABC456ABC") returns 123, and VAL(" 123") also returns 123.

Text Streaming

Text streaming

Topic group

This Help section describes dBL commands that control text streaming to the Command window, a file, or a printer.

Topic group Related topics Example

Outputs the results or return values of one or more expressions to a new line in the Command window results pane. You can also simultaneously stream output to a print buffer and/or text file.

Syntax

```
?
[<exp 1>
 [PICTURE <format expC>]
 [FUNCTION <function expC>]
 [AT <column expN>]
 [STYLE [<fontstyle expN>] or [<fontstyle expC>]]
[,<exp 2>...]
```

<exp 1>[,<exp 2> ...]

Expression(s) of any data type. Output consists of expression results or return values.

PICTURE <format expC>

Formats <exp 1>, or a specified portion of it, with the picture template <format expC>, a character expression consisting of one of the following:

- Template characters.
- Function symbols preceded by @. (You can also use the FUNCTION option, discussed below.)
- Literal characters.
- A combination of template characters, function symbols, and literal characters.
- A variable containing the character expression.

You can use all the template character and function symbols except A, M, R, and S.

FUNCTION <function expC>

Formats all characters in <exp 1> with the function template <function expC>, which must contain one or more function symbols. When you specify function symbols with the FUNCTION option, you don't have to precede them with @.

AT <column expN>

Specifies a character position at which to start the line. The <column expN> argument, effectively a temporary indent, must be between 0 and 255. Note: The AT parameter is ignored if _wrap is *true*.

STYLE [<font expN>] *or* [<style expC>]

Specifies a font number or style for printed output only. Does not apply to file or Command window output (files always use the default printer font, typically Courier, and Command window font style is controlled through Command window properties). See description below for print STYLE specification details.

Description

Use ? to output the results or return values of one or more expressions to a new line in the Command window results pane. You can also simultaneously stream output to a print buffer and/or text file.

Output can be streamed to any or all of these targets at the same time, and streaming to a print buffer or file can be switched on or off any time without affecting the stream to other targets.

You can also stream output using the ?? command. The difference is that ?? appends output to the current line and ? always outputs to a new line.

To use either command, type ? or ?? in the input pane of the Command window (or issue it from a program), follow it with your expression(s) and any optional parameters, then press Enter. The results or return values are immediately streamed to the results pane of the Command window as well as to a print buffer and/or file, if either of those streams is turned on.

To erase the contents of the results pane any time, use the CLEAR command in the input pane. Note that this command only clears the results pane of the Command window and does not affect output to a print buffer or file, if either stream is turned on.

To clear all or a portion of the input pane, select the portion you want to clear and press Del.

Streaming output to a text file

To stream output to a file, first specify a target file with the command SET ALTERNATE TO, e.g.,

```
set alternate to "c:\output.log"
```

A filename is required (no default is supplied).

Then turn on the file output stream with the command SET ALTERNATE ON.

If you specify a file name without an extension, .TXT is appended to the name. If you don’t specify a path, the current path (as shown in the Navigator "Look In" path selector) is used.

You can pause streaming to a file with the command SET ALTERNATE OFF (and resume it again with SET ALTERNATE ON), but to stop streaming and close the file, you must either use CLOSE ALTERNATE or switch output files by issuing another SET ALTERNATE TO command with a different filename or with no filename parameter.

Only one file at a time can be open for text output streaming, and the contents of that open file cannot be viewed until the file is closed using one of the methods above.

If the file you specify in a SET ALTERNATE TO command already exists, you’re given the option of overwriting the existing file. If you choose No, text the named file is not opened for text streaming. In addition, if another file was already open for output, streaming to that file stops and the file is closed. To reopen the closed file, you must reissue SET ALTERNATE TO with the filename; to resume output (append to the previously closed file at the point output was cut off), use ADDITIVE as described below, then reissue the SET ALTERNATE ON command.

If you do choose Yes and overwrite the existing file, the file is immediately emptied. Text streaming will not start again, however, until you issue the SET ALTERNATE ON command.

To append to an existing file, use ADDITIVE with SET ALTERNATE TO:

```
set alternate to "c:\output.log" additive
```

The overwrite warning is not issued when ADDITIVE is used.

Tip

Since ? begins with a line feed to create each new output line, the first line in a new output file or print buffer will be blank. If you want the first line to contain text output instead, use the ?? command for the first item (rather than the ? command).

```
?? "first item"
```

```
? "second item"
```

Streaming output to a printer

To stream output to a print buffer, use SET PRINTER ON. To print the contents of the print buffer to your current printer, use SET PRINTER TO. To print the buffer to a different printer, specify a port or network printer, e.g.,

```
set printer to lpt1
```

or

```
set printer to \\server9\printer4
```

The print buffer continues to receive output until you print the contents of the buffer or issue SET PRINTER OFF.

To format printed output in a particular font, size and style, you can either specify a font in "Font,Size,Style" format or use GETFONT( ) to choose a style. Either way, the font definition is applied to the ? command's STYLE parameter to format the associated line, word or block. This example shows three ways to set the font:

```
// turn on the print buffer
set printer on
// assign a font definition to the variable headingStyle by using getfont( )
headingStyle = getfont( )
? "Program Log" style headingStyle
// you can also specify a font and apply it without using getfont( )
bodyStyle = "Arial,9,Swiss"
// font formatting for the date( ) item in this block
// is specified directly with the STYLE parameter
? "Date: " style "Arial,9,BI,Swiss", date( ) style bodyStyle
// print the two lines to the default printer
set printer to
```

You can get definitions for any font on your system by using GETFONT( ) to open a Font dialog, selecting your font options, and examining the result:

```
s = getfont( )
? s
// selecting 16-pt Arial bold in the Font dialog returns "Arial,16,B,Swiss"
```

To apply a print style to the default font, you can use codes as shown below in place of a font specification:

```
//make default text:
? "bold" style "B"
? "italic" style "I"
? "strikeout" style "S"
? "underlined" style "U"
? "superscript" style "R"
? "subscript" style "L"
? "or use any combination, such as bold italic" style "BI"
```

To overstrike a line of text from a program file in printed output, use the AT option with _wrap set to *false*. To overwrite rather than overstrike text in printed output, use the AT option with _wrap set to *true*, which causes only the second line to print.

To override both an overall _alignment setting and individual paragraph alignments in a memo field, use the B, I, or J functions.

Tip

If SET SPACE is ON (startup default; to test, use ? SET("SPACE")), a space is inserted between multiple expressions streamed out to the same line and between expressions appended to the current line with the ?? command. To remove the spaces for literal formatting, use SET SPACE OFF.

??

Topic group Related topics Example

Appends the value of one or more expressions to the current line in the Command window, print buffer or a file.

Syntax

```
??
[<exp 1>
 [PICTURE <format expC>]
 [FUNCTION <function expC>]
 [AT <column expN>]
 [STYLE [<fontstyle expN>] [<fontstyle expC>]]
[,<exp 2>...]
```

Description

The ?? command is identical to the ? command, except it appends output to the end of the current line rather than to a new line.

???

Topic group Related topics Example

Sends output directly to the printer, bypassing the installed printer driver. This command is provided for compatibility with dBASE IV but is not recommended in *dBASE Plus*

Syntax

??? <expC>

<expC>

A character string to send to the printer.

Description

??? is used in the DOS environment to send printer control codes when the current printer driver does not support a particular printing capability. Printer codes instruct a printer to modify its printing style (italic, bold, and underlined) and page orientation (portrait or landscape).

If you do want to send printer codes with ??? in *dBASE Plus*, test their behavior with the print driver you intend to use. ??? is supported only for very commonly used printers, such as the HP Laser Jet series, and might not be supported in your environment.

In *dBASE Plus*, you can use ? with the STYLE option for font style, and the _porientation system memory variable for page orientation.

CHOOSEPRINTER( )

Topic group Related topics

Opens a printer setup dialog box. Returns *false* if you cancel out of the dialog, *true* otherwise.

Syntax

CHOOSEPRINTER([<title expC>][, <expL>])

<title expC>

Optional custom title for the printer setup dialog box.

<expL>

If *true*, CHOOSEPRINTER( ) will display the "Print Setup" dialog.

If *false*, CHOOSEPRINTER( ) will display the standard "Print" dialog."

Description

Use CHOOSEPRINTER( ) to open a printer setup dialog box, which lets you change the current printer or printer options.

```
p = chooseprinter()
? p
// opens the printer setup dialog; p = false only if the dialog is canceled
chooseprinter ("Tip: For 2-sided printing, see Options, Paper/Output options")
// opens the printer setup dialog with a printing tip in the title
```

If you use CHOOSEPRINTER( ) to switch printers, SET PRINTER TO, _pdriver, _plength, and _porientation will automatically point to the new printer.

To activate a specific printer driver, you can also use _pdriver.

Menu equivalent: File | Print opens the Print dialog, which offers a printer selection list and properties dialog for the selected printer.

The CHOOSEPRINTER( ) function is maintained only for backward compatibility. We suggest using the printer object for the _app or reports.

```
_app.printer.choosePrinter() //sets the default printer
report.printer.choosePrinter()//sets the printer for that instance of the report
```

CLEAR

Topic group

Clears the Command window results pane.

Syntax

CLEAR

Description

Use CLEAR to remove the contents of the results pane of the Command window.

CLOSE ALTERNATE

Topic group Related topics

Close the text stream file opened with SET ALTERNATE

Syntax

CLOSE ALTERNATE

Description

CLOSE ALTERNATE is equivalent to issuing SET ALTERNATE TO with no filename. See SET ALTERNATE for details.

CLOSE PRINTER

Topic group Related topics

Close the print buffer, sending buffered output to the printer.

Syntax

CLOSE PRINTER

Description

CLOSE PRINTER is equivalent to issuing SET PRINTER TO with no options. See SET PRINTER for details.

EJECT

Topic group Related topics Example

Advances printer paper to the top of the next page.

Syntax

EJECT

Description

Use EJECT to position printed output on the page. If you are using a tractor-feed printer (such as a dot matrix printer) and the paper is correctly positioned, EJECT advances the paper to the top of the next sheet. If you are using a single-sheet printer (such as a laser printer), EJECT prints any data in the print queue and ejects the page. Before printing or executing EJECT, connect and turn on the printer.

EJECT works in conjunction with _padvance, _plength, and _plineno. If _padvance is set to "FORMFEED" (the default), issuing the EJECT command from *dBASE Plus* is equivalent to using your printer's formfeed button or sending the formfeed character (ASCII 12) to the printer. If _padvance is set to "LINEFEEDS", issuing EJECT sends individual linefeeds to the printer until _plineno equals _plength, then resets _plineno to 0. Then, _pageno is incremented by 1. For more information, see _padvance.

EJECT is often used in when printing reports. For example, if PROW( ) returns a value that is close to the bottom of the page, issue EJECT to continue the report at the top of the next page. EJECT automatically resets the printhead to the top left corner of the new page, which is where PROW( ) = 0 and PCOL( ) = 0.

EJECT is the same as EJECT PAGE, except EJECT PAGE also executes any page-handling routine you've defined with ON PAGE.

EJECTPAGE

Topic group Related topics Example

Advances printer paper to the top of the next page and executes any ON PAGE command.

Syntax

EJECT PAGE

Description

Use EJECT PAGE with ON PAGE to control the ejection of pages by a printer. If you define a page-handling routine with ON PAGE AT LINE <expN> and then issue EJECT PAGE, *dBASE*

Plus checks to see if the current line number (_plineno) is greater than the line number specified by <expN>. If _plineno is less than the ON PAGE line, EJECT PAGE sends sufficient linefeeds to trigger the ON PAGE page-handling routine.

If _plineno is greater than the ON PAGE line, or if you don't have an ON PAGE page-handling routine, EJECT PAGE advances the output as follows:

If _padvance is set to "FORMFEED" and SET PRINTER is ON, dBASE issues a formfeed (ASCII code 12).

If _padvance is set to "LINEFEEDS" and SET PRINTER is ON, dBASE issues sufficient linefeeds (ASCII code 10) to advance to the next page. It uses the formula _plength – _plineno to calculate the number of linefeeds.

If you direct output to a destination other than the printer (for example, if you use SET ALTERNATE or SET DEVICE), dBASE uses the formula _plength – _plineno to calculate the number of linefeeds.

After ejecting a page, EJECT PAGE increments _pageno by 1 and resets _plineno to 0.

ON PAGE

Topic group Related topics Example

Executes a specified command when printed output reaches a specified line on the current page.

Syntax

ON PAGE
[AT LINE <expN> <command>]

AT LINE <expN>

Identifies the line number at which to execute the specified page-formatting command.

<command>

The command to execute when printed output reaches the specified line number, <expN>. To execute more than one command, issue ON PAGE DO <filename>, where <filename> is a program or procedure file containing the sequence of commands to execute.

Description

Use ON PAGE to specify a command to execute when printed output reaches a specific line number. ON PAGE with no options disables any previous ON PAGE statement.

The value of the _plineno system variable indicates the number of lines that have been printed on the current page. As soon as the _plineno value is equal to the value you specify for <expN>, *dBASE Plus* executes the ON PAGE command.

Use the ON PAGE command to print headers and footers. For example, the on page command can call a procedure when the _plineno system memory variable reaches the line number that signifies the end of a page. In turn, that procedure can call two procedures, one to print the footer on the current page and one to print the header on the next page.

You can begin header routines with EJECT PAGE to ensure that the header text prints at the top of the following page. EJECT PAGE also sets the _plineno system memory variable to 0. Use the ? command at the beginning of a header procedure to skip several lines before printing the header information. You can also use the ? command at the end of the procedure to skip several lines before printing the text for the page.

Begin footer routines with the ? command to move several lines below the last line of text. You can use the ?? command with the _pageno system memory variable to print a page number for each page on the same line as the footer.

To calculate the appropriate footer position, add the number of lines for the bottom margin and the number of lines for the footer text to get the total lines for the bottom of the page. Subtract this total from the total number of lines per page. Use this result to specify a number for the AT LINE argument. If the footer text exceeds the number of lines per page, the remainder prints on the next page.

PCOL( )

Topic group Related topics Example

Returns the printing column position of a printer. Column numbers begin at 0.

Syntax

PCOL( )

Description

Use PCOL( ) to determine the horizontal printing position of a printer—that is, the column at which the printer is set to begin printing. Use PCOL( ) in mathematical statements to direct the printer to begin printing at a position relative to its current column position. For example, PCOL( ) + 5 represents a position five columns to the right of the current position, and PCOL( ) – 5 represents a position five columns to the left of the current position.

When you direct output to the printer, *dBASE Plus* maps each character according to the coordinate plane, a two-dimensional grid. The coordinate plane is divided into character cells whose widths depend on the value of _ppitch. See the table in the description of _ppitch, which lists _ppitch values. The height of each character cell is determined by the size of the font of the parent form window.

PCOL( ) returns a column number that reflects the current value of _ppitch, regardless of whether you're printing with proportional or monospaced fonts. If you're printing with a proportional font, you can add and subtract fractional numbers to and from the PCOL( ) value to move the printing position accurately.

SET PRINTER must be ON for PCOL( ) to return a column position; otherwise, it returns 0.

PRINTJOB...ENDPRINTJOB

Topic group Related topics Example

Uses the values stored in system memory variables to control a printing operation.

Syntax

PRINTJOB
<statements>
ENDPRINTJOB

<statements>

Any valid dBL statements.

Description

Use PRINTJOB...ENDPRINTJOB to control a printing operation with the values of the system memory variables _pbpage, _pepage, _pcopies, _peject, and _plineno. When *dBASE Plus* begins executing PRINTJOB, it does the following:

1. Closes the current print document (if any) and begins a new one, as if you had issued CLOSE PRINTER before issuing PRINTJOB
2. Ejects a page if _peject is set to "BEFORE" or "BOTH"
3. Sets _pcolno to 0

When *dBASE Plus* reaches ENDPRINTJOB, it does the following:

4. Ejects a page if _peject is set to "AFTER" or "BOTH"
    - Resets _pcolno to 0

Before using PRINTJOB...ENDPRINTJOB, set the relevant system memory variables and issue SET PRINTER ON. After ENDPRINTJOB, use CLOSE PRINTER to close and print the document.

PRINTSTATUS( )

Topic group Related topics Example

Returns *true* if the print device is ready to accept output.

Syntax

PRINTSTATUS([<port name expC>])

<port name expC>

A character expression such as "lpt1" that identifies the printer port to check.

Description

Use PRINTSTATUS( ) to determine whether you've designated a printer port as an output device with SET PRINTER TO <port name expC>. In *dBASE Plus*, the Windows Print Manager spools print output to and manages the printer port. Therefore, the Print Manager informs you when a printer isn't ready to receive output.

If you don't pass <port name expC> to PRINTSTATUS( ), it checks the default port you specified with SET PRINTER TO. PRINTSTATUS( ) returns only *false* if you haven't specified a printer port with SET PRINTER TO or if the port you specify hasn't been set with SET PRINTER TO.

Note

dBASE Plus automatically executes SET PRINTER TO on startup if the WIN.INI file contains a valid printer definition. See your Windows documentation for information on WIN.INI settings.

PROW( )

Topic group Related topics

Returns the printing row position of a printer. Row numbers begin at 0.

Syntax

PROW( )

Description

Use PROW( ) to determine the vertical printing position of a printer—that is, the row at which the printer is set to begin printing. Use PROW( ) in mathematical statements to direct the printer to begin printing at a position relative to its current row position. For example, PROW( ) + 5 represents a position five rows below the current position and PROW( ) – 5 represents a position five rows above the current position.

When you direct output to the printer, *dBASE Plus* maps each character according to the coordinate plane, a two-dimensional grid. The coordinate plane is divided into character cells whose widths depend on the value of _ppitch. See the table in the description of _ppitch, which lists _ppitch values. The height of each character cell is determined by the size of the font of the parent form window.

If you're printing with a proportional font, you can add and subtract fractional numbers to and from the PROW( ) value to move the printing position accurately. If you issue ? without the STYLE option and use only integer coordinates, *dBASE Plus* uses the default printer font (typically Courier or another monospaced font).

SET PRINTER must be ON for PROW( ) to return a row position; otherwise, it returns 0.

SET ALTERNATE

Topic group Related topics Example

Controls the recording of input and output in an alternate text file.

Syntax

SET ALTERNATE on | OFF

SET ALTERNATE TO [<filename> | ? | <filename skeleton> [ADDITIVE]]

<filename> | ? | <filename skeleton>

The alternate text file, or target file, to create or open. The ? and <filename skeleton> options display a dialog box in which you can specify a new file or select an existing file. If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, *dBASE Plus* assumes .TXT.

ADDITIVE

Appends *dBASE Plus* output that appears in the results pane of the Command window to the specified existing alternate file. If the file doesn't exist, *dBASE Plus* returns an error message.

Default

The default for SET ALTERNATE is OFF. To change the default, set the ALTERNATE parameter in the [OnOffSetting Settings] section of PLUS.ini. To set a default file name for use with SET ALTERNATE, specify an ALTERNATE parameter in the [CommandSettings] section of PLUS.ini.

Description

Use SET ALTERNATE TO to create a record of *dBASE Plus* output and commands. You can edit the contents of this file with the Text Editor for use in documents, or store it on disk for future reference. You can record, edit, and incorporate command sequences into new programs.

SET ALTERNATE TO <filename> only opens an alternate file, while SET ALTERNATE ON | OFF controls the storage of input and output to that file. Only one alternate file can be open at a time. When you issue SET ALTERNATE TO <filename> to open a new file, *dBASE Plus* closes the previously open alternate file.

When SET ALTERNATE is ON, *dBASE Plus* stores output to the results pane of the Command window in the text file you've opened by previously issuing SET ALTERNATE TO <filename>. An alternate file must be open for SET ALTERNATE ON to have an effect. SET ALTERNATE

doesn't affect a program's output; it only determines when that output is saved in the alternate file. (Keyboard entries in the Command window aren't stored to the alternate file.)

To prevent your text file from beginning with a blank line, use two question marks (??) before the first word that you send to the alternate file.

Issuing SET ALTERNATE OFF does not close the alternate file. Before accessing the contents of an alternate file, formally close it with CLOSE ALTERNATE or SET ALTERNATE TO (with no file name). This ensures that all data recorded by *dBASE Plus* for storage in the alternate file is transferred to disk, and automatically turns SET ALTERNATE to OFF.

If SET SAFETY is ON and you don't use the ADDITIVE option, and a file exists with the same name as the target file, *dBASE Plus* displays a dialog box asking if you want to overwrite the file. If SET SAFETY is OFF and you don't use the ADDITIVE option, any existing file with the same name as the target file is overwritten without warning.

SET CONSOLE

Topic group Example

Controls the display of output in the results pane of the Command window during program execution.

Syntax

SET CONSOLE ON | off

Description

When SET CONSOLE is ON, *dBASE Plus* displays all text stream output in the results pane of the Command window. Use SET CONSOLE OFF to disable this output. For example, if you are creating a text file with SET ALTERNATE, you usually do not need to see the output in the Command window at the same time. By using SET CONSOLE OFF, your program will execute faster.

Whenever the input pane of the Command window gets focus, SET CONSOLE is always turned ON. The SET CONSOLE command has no effect when issued in the Command window.

You may use the WAIT command while SET CONSOLE is OFF; however, *dBASE Plus* displays neither the prompt for the input nor the input itself.

SET MARGIN

Topic group Related topics Example

Sets the width of the left border of a printed page.

Syntax

SET MARGIN TO <expN>

<expN>

The column number at which to set the left margin. The valid range is 0 to 254, inclusive. You can specify a fractional number for <expN> to position output accurately with a proportional font.

Default

The default for SET MARGIN is 0. To change the default, set the MARGIN parameter in PLUS.ini.

Description

Use SET MARGIN to adjust the printer offset for the left margin for all printed output. The margin established by SET MARGIN becomes the printer's column 0 position. set margin resets the value of the _ploffset system memory variable but doesn't affect the value of the _lmargin system memory variable.

Use SET MARGIN to adjust the position of text on the printed page according to the type of paper. For example, if you're printing to three-hole paper, you might need to increase the left border. You can also use SET MARGIN to compensate for the placement of paper in the printer. For example, if the paper is off-center in the printer, you can adjust the width of the left border to properly place the text.

When you direct output to the printer, *dBASE Plus* maps each character according to the coordinate plane, a two-dimensional grid. The coordinate plane is divided into character cells whose widths depend on the value of _ppitch. See the table in the description of _ppitch, which lists _ppitch values. The height of each character cell is determined by the size of the font.

If you change the value of SET MARGIN, *dBASE Plus* takes the current value of _ppitch into consideration when calculating the cell width of the coordinate plane. This happens regardless of whether you're printing with proportional or monospaced fonts.

SET PCOL

Topic group Related topics Example

Sets the printing column position of a printer, which is the value of PCOL( ).

Syntax

SET PCOL TO <expN>

<expN>

The column number to which to set PCOL( ). The valid range is 0 to 32,767, inclusive.

Description

Use SET PCOL to set the horizontal printing position of a printer, which is the value the PCOL( ) function returns. Generally, you use the command SET PCOL TO 0 to reset the printer column to the left edge of the page.

When you move the printing position to a new line, *dBASE Plus* reinitializes PCOL( ) to 0, so SET PCOL affects the value of PCOL( ) for the current line only. When you send output to your printer, *dBASE Plus* updates PCOL( ) by adding 1 to the current PCOL( ) value for each character it sends to the printer. The printing position moves one column for each character the printer prints.

When you send a printer control code or escape sequence to your printer, the printing position doesn't move. (Printer control codes and escape sequences are strings that give the printer instructions, such as to print underlining, boldface type, or different fonts.) Although control codes and escape sequences don't move the printing position, *dBASE Plus* nonetheless increments the PCOL( ) value by the number of characters that you send to the printer. Each control code character increments the value of PCOL( ) by 1 just like any other character. As a result, the value of PCOL( ) might not reflect the actual printing position. Use SET PCOL to reset the value of PCOL( ) to the same value as the printing position.

To send a control code to the printer without changing the value of PCOL( ), save the current value of PCOL( ) to a memory variable, send the control code to the printer, then SET PCOL to the contents of the memory variable.

SET PRINTER

Topic group Related topics Example

The SET PRINTER TO setting specifies a file to receive streaming output, or uses a device code recognized by the Windows Print Manager to designate a printer. The On/Off setting controls whether *dBASE Plus* also directs streaming output that appears in the Command window to the device or file specified by SET PRINTER TO.

Syntax

SET PRINTER on | OFF

SET PRINTER TO [<filename> | ? | <filename skeleton>] | [<device>]

<filename> | ? | <filename skeleton>

The text file to send output to instead of the printer. By default, *dBASE Plus* assigns a .PRT extension to <filename> and saves the file in the current directory. The ? and <filename skeleton> options display a dialog box, in which you specify the name of the target file and the directory to save it in.

<device>

The printer port of the printer to send output to. Specify printers and their ports with the Windows Control Panel.

Default

The default for SET PRINTER is OFF. To change the default, set the PRINT parameter in the [OnOffCommandSettings] section in PLUS.ini. The default for SET PRINTER TO is the default printer you specify with the Windows Control Panel.

Description

Use SET PRINTER TO to direct streaming output from commands such as ?, ??, and LIST to a printer or a text file. SET PRINTER TO with no option sends this output to the default printer.

Use SET PRINTER ON/OFF to enable or disable the printer you specify with SET PRINTER TO.

To send streaming output to a file rather than the printer, issue SET PRINTER TO FILE <filename>. When you issue SET PRINTER TO FILE <filename>, issuing SET PRINTER ON directs streaming output to the text file <filename> rather than to the printer. The file has the default extension of .PRT.

When SET PRINTER is OFF, *dBASE Plus* directs streaming output only to the result pane of the Command window. SET PRINTER must be ON to output data to a text file unless you issue a command with its TO PRINTER option. The following example illustrates this behavior:

```
set printer off
set printer to file test.prt
type file.txt // displays on screen only
type file.txt to print // output sent to screen and test.prt
```

SET PROW

Topic group Related topics

Sets the current row position of a printer's print head, which is the value of PROW( ).

Syntax

SET PROW TO <expN>

<expN>

The row number to which to set PROW( ). The valid range is 0 to 32,767, inclusive.

Description

Use SET PROW to set the vertical printing position of a printer, which is the value the PROW( ) function returns. Generally, you use the command SET PROW TO 0 to reset the printer row to top-of-page.

SET SPACE

Topic group Related topics Example

Determines whether *dBASE Plus* inserts a space between expressions displayed or printed with a single ? or ?? command.

Syntax

SET SPACE ON | off

Default

The default for SET SPACE is ON. To change the default, set the SPACE parameter in PLUS.ini.

Description

Use SET SPACE OFF when you use a single ? or ?? command to print a list of expressions and you don't want spaces between the expressions. If you want the expressions printed with spaces between them, issue SET SPACE ON.

SET SPACE has no effect on multiple ? or ?? commands. For example, if you issue the command ?? <exp> twice, the second instance of <exp> will be printed adjacent to the first, even if SET SPACE is ON. However, if SET SPACE is ON and you issue ?? <exp>, <exp> as a single command, there will be a space between the two instances of <exp>.

_alignment

Topic group Related topics Example

Left-aligns, right-aligns, or centers ? and ?? command output within margins specified by _lmargin and _rmargin when _wrap is true.

Syntax

_alignment = <expC>

<expC>

The character expression "LEFT", "CENTER", or "RIGHT". You can enter <expC> in any combination of uppercase and lowercase letters.

Default

The default for _alignment is "LEFT".

Description

Use _alignment to left-align, right-align, or center output from the ? and ?? commands between the margins you set with _lmargin and _rmargin. The _alignment setting is effective only when _wrap is true (*true*).

To control the alignment of text within a field, use the "B," "I," and "J" format options with the PICTURE or FUNCTION options.

_indent

Topic group Related topics Example

Specifies the number of columns to indent the first line of a paragraph of ? command output when _wrap is true.

Syntax

_indent = <expN>

<expN>

The column number, relative to the left margin, where the first line of a new paragraph begins. You can specify a fractional number for <expN> to position output accurately with a proportional font.

Default

The default for _indent is 0.

Description

Use _indent to specify where the first line of a new paragraph begins relative to the left margin. (Specify the left margin with _lmargin.) The _indent setting is effective only when _wrap is *true*.

When you direct ? output to the printer, *dBASE Plus* maps each character according to the coordinate plane, a two-dimensional grid. The coordinate plane is divided into character cells whose widths depend on the value of _ppitch. See the table in the description of _ppitch, which lists _ppitch values. The height of each character cell is determined by the size of the font.

If you change the value of _indent, *dBASE Plus* takes the current value of _ppitch into consideration when calculating the cell width of the coordinate plane. This happens regardless of whether you're printing with proportional or monospaced fonts.

To indent the first line of a paragraph, use a value greater than 0. For example, to begin the line five columns to the right of the left margin, set _indent to 5. To create a hanging indent (sometimes called an outdent), use a negative value. For example, to begin the first line five columns to the left of the left margin, set _indent to -5. Using the default value of 0 (no indent or outdent) aligns all lines in a paragraph to the left margin. The sum of _lmargin and _indent must be greater than 0 and less than _rmargin.

_lmargin

Topic group Related topics Example

Defines the left margin for ? and ?? command output when _wrap is true.

Syntax

_lmargin = <expN>

<expN>

The column number of the left margin. The valid range is 0 to 254, inclusive. You can specify a fractional number for <expN> to position output accurately with a proportional font.

Default

The default for _lmargin is 0.

Description

Use _lmargin to set the left margin for ? and ?? command output. If you're sending output to a printer, _lmargin sets the left margin from the _ploffset (page left offset) column. For example, if _ploffset is 10 and _lmargin is 5, output prints from the 15th column. The _lmargin setting is effective only when _wrap is *true*.

When you direct output to the printer, *dBASE Plus* maps each character according to the coordinate plane, a two-dimensional grid. The coordinate plane is divided into character cells whose widths depend on the value of _ppitch. See the table in the description of _ppitch, which lists _ppitch values. The height of each character cell is determined by the size of the font.

If you change the value of _lmargin, *dBASE Plus* takes the current value of _ppitch into consideration when calculating the cell width of the coordinate plane. This happens regardless of whether you're printing with proportional or monospaced fonts.

If you use _indent to specify the indentation of the first line of each paragraph, the combined values of _lmargin and _indent must be less than the value of _rmargin.

_padvance

Topic group Related topics Example

Determines whether the printer advances the paper of a print job with a formfeed or with linefeeds.

Syntax

_padvance = <expC>

<expC>

The character expression "FORMFEED" or "LINEFEEDS".

Default

The default for _padvance is "FORMFEED".

Description

Use _padvance to specify whether *dBASE Plus* advances the paper to the top of the next sheet one sheet at a time using a formfeed character, or one line at a time using linefeed characters. If you use the default "FORMFEED" setting, the paper advances according to the printer's default form length setting.

Tractor-feed printers (such as dot matrix printers) generally use a "LINEFEEDS" setting, while form feed printers (such as laser printers) generally use a "FORMFEED" setting.

Note

Sending CHR(12) to the printer always issues a formfeed, even if you set _padvance to "LINEFEEDS".

Use the "LINEFEEDS" setting if you change the length of the paper or want to print a different number of lines than the default form length of the printer without adjusting its setting. For

example, to print short pages, such as checks that are 20 lines long, set _plength to the length of the output (20 in this example) and _padvance to "LINEFEEDS."

The number of linefeeds *dBASE Plus* uses to reach the top of the next page depends on whether you issue an eject during streaming or non-streaming output mode.

An eject occurs during streaming output mode when you issue:

EJECT PAGE without an ON PAGE handler
EJECT PAGE with an ON PAGE handler when the current line position is past the ON PAGE line
PRINTJOB or ENDPRINTJOB and _peject causes an eject

In these cases, *dBASE Plus* calculates the number of linefeeds to send to the print device using the formula _plength – _plineno.

An eject occurs in nonstreaming output mode when you issue:

EJECT

In these cases, *dBASE Plus* calculates the number of linefeeds to send to the print device using the formula _plength – MOD(PROW( ), _plength).

_pageno

Topic group Related topics Example

Determines or sets the current page number.

Syntax

_pageno = <expN>

<expN>

An integer from 1 to 32,767, inclusive.

Description

Use _pageno to number pages of streaming output from commands such as ?, ??, and LIST.

With _pageno, you can determine the current page number or set the page number to a specific value. Use it to print page numbers in a report or, when combining documents, to assign an incremented number to the first page of the second document.

A page break occurs when the value of _plineno (the line number count) becomes greater than the value of _plength (the currently defined printed page length in lines). At each page break of streaming output, *dBASE Plus* automatically increments the value of _pageno.

_pbpage

Topic group Related topics Example

Specifies the page number of the first page PRINTJOB prints.

Syntax

_pbpage = <expN>

<expN>

The page number at which to begin printing. The valid range is 1 to 32,767, inclusive. Specify a positive integer for <expN>.

Default

The default for _pbpage is 1.

Description

Use _pbpage to begin printing a print job at a specific page number. Pages with numbers less than _pbpage don't print. To stop printing at a specific page number, use _pepage.

If you set _pbpage to a value greater than _pepage, *dBASE Plus* returns an error.

_pcolno

Topic group Related topics Example

Identifies or sets the current column number of streaming output.

Syntax

_pcolno = <expN>

<expN>

The column number at which to begin printing. The valid range is 0 to 255, inclusive. You can specify a fractional number for <expN> to position output accurately with a proportional font.

Default

The default for _pcolno is 0.

Description

Use _pcolno to position printing streaming output from commands such as ?, ??, and LIST.

When you direct output to the printer, *dBASE Plus* maps each character according to the coordinate plane, a two-dimensional grid. The coordinate plane is divided into character cells whose widths depend on the value of _ppitch. See the table in the description of _ppitch, which lists _ppitch values. The height of each character cell is determined by the size of the font.

If you change the value of _pcolno, *dBASE Plus* takes the current value of _ppitch into consideration when calculating the cell width of the coordinate plane. This happens regardless of whether you're printing with proportional or monospaced fonts.

The PCOL( ) function also returns the current printhead position of the printer, but if SET PRINTER is OFF, the PCOL( ) value doesn't change. _pcolno, on the other hand, returns or assigns the current position in the streaming output regardless of the SET PRINTER setting.

_pcopies

Topic group Related topics Example

Specifies the number of copies to print for a PRINTJOB.

Syntax

_pcopies = <expN>

<expN>

The number of copies to print. The valid range is 1 to 32,767, inclusive. Specify a positive integer for <expN>.

Default

The default for _pcopies is 1.

Description

Use _pcopies to print a specific number of copies of a print job. You can assign a value to _pcopies in the Command window or in a program. The value of _pcopies has an effect only when you send a print job to the printer by issuing PRINTJOB. In a program, assign a value to _pcopies before issuing PRINTJOB.

_pdriver

Topic group Related topics Example

Identifies the current printer driver or activates a new driver.

Syntax

_pdriver = <expC>

<expC>

The name of the printer driver to activate.

Default

The default for _pdriver is the printer driver you specify with the Windows Control Panel. If you haven't specified a printer driver with the Control Panel, the value of _pdriver is an empty string ("").

Description

Use _pdriver to identify the current printer driver or to activate an installed driver. (To install a new printer driver, use the Windows Control Panel.)

The _pdriver value contains two elements separated by a comma: the base file name of the Windows driver file and the name of the printer as it appears in WIN.INI. The current driver might not identify a printer name, in which case, _pdriver contains only the driver file name. For example, if the current printer driver is for the HP Laserjet IIISi PostScript printer, _pdriver may contain the value "pscript,HP LaserJet IIISi PostScript". To activate this driver, issue the command _pdriver = "pscript,HP LaserJet IIISi PostScript".

To activate a driver from within *dBASE Plus*, it may be easier to use CHOOSEPRINTER( ) than to assign a value to _pdriver. To activate a driver in Windows, use the Printers program of the Windows Control Panel. CHOOSEPRINTER( ) opens the Print Setup dialog box, in which you can also select options such as paper size, source, and orientation (portrait or landscape). In the Windows Control Panel, you can choose Setup to select these options.

_pecode

Topic group Related topics

System variable initialized to an empty string.

Syntax

_pecode = <expC>

<expC>

Character expression up to 255 characters.

Default

Empty string.

Description

During execution of a ENDPRINTJOB command dBASE will send the contents of _PECODE to the printer..

_peject

Topic group Related topics Example

Determines whether *dBASE Plus* ejects a sheet of paper before and after a PRINTJOB.

Syntax

_peject = <expC>

<expC>

The character expression "before", "after", "both", or "none".

Default

The default for _peject is "before", which tells the printer to eject a sheet of paper before starting the print job.

Description

Use _peject to specify if and when the printer should eject a sheet of paper. Assign a new value to _peject (and to any other system memory variable) before issuing PRINTJOB in a program to make the new value affect the print job.

The following table describes _peject options.

<expC>	Result
"before"	Eject sheet before printing the first page
"after"	Eject sheet after printing the last page
"both"	Eject sheet before and after the print job
"none"	Don't eject sheet before or after the print job

Note

The _peject system memory variable is distinct from the EJECT command, which tells the printer to advance the paper to the top of the next page.

_pepage

Topic group Related topics Example

Specifies the page number of the last page of a print job.

Syntax

_pepage = <expN>

<expN>

The page number of the last page to print. The valid range is 1 to 32,767, inclusive. You must specify a positive integer for <expN>.

Default

The default for _pepage is 32,767.

Description

Use _pepage to stop printing a print job at a specific page number. Pages with numbers greater than _pepage don't print. To begin printing at a specific page number, use _pbpage.

If you set _pepage to a value less than _pbpage, *dBASE Plus* returns an error.

_pform

Topic group Related topics Example

Identifies the current print form file or activates another one.

Syntax

_pform = <filename>

<filename>

The name of a print form file (.PRF).

Default

The default for _pform is an empty string ("").

Description

Use _pform to determine the name of the current print form file or to activate another one. A print form file (.PRF) is a binary file that contains print settings for printing a print job. The print form file contains the following system memory variables:

Variable	Action
_padvance	Determines whether the printer advances the paper with a formfeed or linefeeds.
_pageno	Determines or sets the current page number.
_pbpage	Specifies the page number at which PRINTJOB begins printing.
_pcopies	Specifies the number of copies to print in a printjob.
_pdriver	Activates a specified printer driver. (If the print form file is from dBASE IV, *dBASE Plus* ignores this value.)
_peject	Controls page ejects before and after PRINTJOB.
_pepage	Specifies the number of the last page that PRINTJOB prints.
_plength	Specifies the number of lines per page for streaming output.
_ploffset	Determines the width of the left border on a printed page.
_ppitch	Sets the printer pitch, the number of characters per inch that the printer prints.
_pquality	Specifies if the printer prints in letter-quality or draft mode.
_pspacing	Sets the line spacing for streaming output.

When you specify a print form file by assigning its name to _pform, the values stored in the file are assigned to their respective variables. Jobs you send to the printer then behave in accordance with these variables.

_plength

Topic group Related topics Example

Specifies the number of lines per page for streaming output.

Syntax

_plength = <expN>

<expN>

The number of lines per page. The valid range is 1 to 32,767, inclusive. You can specify a fractional number for <expN> to position output accurately with a proportional font.

Default

The default page length is determined by the default page size of the current printer driver and the current page orientation (portrait or landscape).

Description

Use _plength to specify a page length that is different from the default of the current printer driver. For example, to print short pages, such as checks that are 20 lines long, set _plength to the length of the output (20 in this example) and _padvance to "LINEFEEDS" to advance to the top of the next page.

When you change printer drivers or page orientation, *dBASE Plus* changes the value of _plength automatically. You can change printer drivers in *dBASE Plus* by issuing CHOOSEPRINTER( ) or by assigning a value to _pdriver. In Windows, you can change printer drivers with the Printers program of the Control Panel (however, it won't take effect until you quit *dBASE Plus* and start a new *dBASE Plus* session). You can change page orientation in any of these ways or, in *dBASE PLus*, by changing the value of _porientation.

_plineno

Topic group Related topics Example

Identifies or sets the current line number of streaming output.

Syntax

_plineno = <expN>

<expN>

The line number at which to begin printing. The valid range is 0 to _plength – 1. You can specify a fractional number for <expN> to position output accurately with a proportional font.

Default

The default for _plineno is 0.

Description

Use _plineno to position printing streaming output from commands such as ?, ??, and LIST.

The PROW( ) function also returns the current printhead position of the printer, but if SET PRINTER is OFF, the PROW( ) value doesn't change. _plineno, on the other hand, returns or assigns the current position in the streaming output regardless of the SET PRINTER setting.

_ploffset

Topic group Related topics Example

Displays or sets the width of the left border of a printed page.

Syntax

_ploffset = <expN>

<expN>

The column number at which to set the left margin. The valid range is 0 to 254, inclusive. You can specify a fractional number for <expN> to position output accurately with a proportional font.

Default

The default for _ploffset is 0. To change the default, set the MARGIN parameter in Plus.ini.

Description

Use _ploffset (page left offset) to specify the distance from the left edge of the paper to the left margin of the print area. Use _lmargin to set the left margin from the _ploffset column. For example, if _ploffset is 10 and _lmargin is 5, output prints from the 15th column.

The _ploffset system memory variable is equivalent to the SET MARGIN value. Changing the value of one changes the other. For more information, see SET MARGIN.

_porientation

Topic group Related topics Example

Determines whether the printer prints in portrait or landscape mode.

Syntax

_porientation = <expC>

<expC>

The character expression "PORTRAIT" or "LANDSCAPE".

Default

The default for _porientation is the orientation you specify with the Printers program of the Windows Control Panel or, in *dBASE Plus*, with the CHOOSEPRINTER( ) function. By default, this orientation is portrait.

Description

Use _porientation to specify whether you want to print in portrait or landscape mode. When you print in portrait mode, each page is read vertically; a standard American letter-size piece of paper is 8.5 inches wide by 11 inches long. When you print in landscape mode, each page is read horizontally; a standard American letter-size piece of paper is 11 inches wide by 8.5 inches long.

Most printer drivers support landscape printing; however, if you specify landscape while using a printer driver that doesn't support it, the printer continues to print in portrait mode, possibly truncating text.

Changing page orientation automatically resets _plength.

If _porientation is changed during printing, it will only take effect when _plineno = 0 or after a page eject occurs. Since _plineno may be greater than 0 when a print routine begins, you should set _plineno = 0 before attempting to change the orientation of a page.

_ppitch

Topic group Related topics Example

Sets the printer pitch, the number of characters per inch that the printer prints.

Syntax

_ppitch = <expC>

<expC>

The character expression "pica", "elite", "condensed", or "default".

Default

The default for _ppitch is "default", the pitch defined by your printer's settings or by setup codes or commands you sent to the printer before you started *dBASE Plus*. "Default" means that *dBASE Plus* hasn't sent any pitch control codes to the printer.

Description

Use _ppitch to set the pitch (characters per inch) on the printer. The _ppitch setting sends a control code appropriate to the current printer driver. Use the Windows Control Panel or CHOOSEPRINTER( ) to select the printer driver.

When you direct output to the printer, *dBASE Plus* maps each character according to the coordinate plane, a two-dimensional grid. The coordinate plane is divided into character cells whose sizes depend on the value of _ppitch. The height of each character cell is determined by the size of the font.

The following table lists _ppitch values.

_ppitch value	Character cell width
"pica"	1/10" (10 characters/inch)
"elite"	1/12" (12 characters/inch)
"condensed"	1/17" (17 characters/inch)

If you change the value in other system memory variables such as _lmargin, _rmargin, and _ploffset, *dBASE Plus* takes the current value of _ppitch into consideration when calculating the cell width of the coordinate plane. This happens regardless of whether you're printing with proportional or monospaced fonts.

_pquality

Topic group Related topics Example

Specifies whether the printer prints in letter-quality or draft mode. Used primarily with dot-matrix printers; the _pquality value usually has no effect on printers that don't support draft mode, such as laser and Postscript printers.

Syntax

_pquality = <expL>

<expL>

The logical expression *true* for letter quality and *false* for draft quality.

Default

The default for _pquality is *false* for draft mode.

Description

Use _pquality to determine whether the printer prints in letter-quality or draft mode. Letter-quality mode produces printed copy of higher quality (finer resolution) than draft; however, draft mode usually prints more quickly than letter-quality, depending on the printer.

_pscode

Topic group Related topics

System variable initialized to an empty string.

Syntax

_pscode = <expC>

<expC>

Character expression up to 255 characters.

Default

Empty string.

Description

During execution of a PRINTJOB command dBASE will send the contents of _PSCODE to the printer..

_pspacing

Topic group Related topics Example

Sets the line spacing for streaming output.

Syntax

_pspacing = <expN>

<expN>

The amount of line spacing. The valid range is 1 to 3, inclusive:

A value of 1 represents single spacing.
A value of 2 represents double spacing. There is one blank line between printed lines.
A value of 3 represents triple spacing. There are two blank lines between printed lines.

Paragraph spacing is in multiples of the height of the line just printed, which depends on the tallest font used in printing the line. You can specify a fractional number for <expN> to space text by partial line heights.

Default

The default for _pspacing is 1, which sets line spacing to single-line.

Description

Use _pspacing to set the line spacing of streaming output from commands such as ?, ??, and LIST. To insert a single blank line into output, use the ? command.

_rmargin

Topic group Related topics Example

Defines the right margin for ? and ?? command output when _wrap is true.

Syntax

_rmargin = <expN>

<expN>

The column number of the right margin. The valid range is 0 to 255, inclusive. You can specify a fractional number for <expN> to position output accurately with a proportional font.

Default

The default for _rmargin is 79.

Description

Use _rmargin to set the right margin for output from the ? and ?? commands. The value of _rmargin must be greater than the value of _lmargin or _lmargin + _indent. For example, if _lmargin and _indent are both set to 5, _rmargin must be greater than 10 to display at least one column of output.

When you direct output to the printer, *dBASE Plus* maps each character according to the coordinate plane, a two-dimensional grid. The coordinate plane is divided into character cells whose widths depend on the value of _ppitch. See the table in the description of _ppitch, which lists _ppitch values. The height of each character cell is determined by the size of the font.

If you change the value of _rmargin, *dBASE Plus* takes the current value of _ppitch into consideration when calculating the cell width of the coordinate plane. This happens regardless of whether you're printing with proportional or monospaced fonts.

_tabs

Topic group Related topics Example

Sets one or more tab stops for output from the ? and ?? commands.

Syntax

_tabs = <expC>

<expC>

The list of column numbers for tab stops. If you set more than one tab stop, the numbers must be in ascending order and separated by commas. Enclose the entire list in quotation marks. You can specify fractional numbers for <expC> to position output accurately with a proportional font.

Default

The default for _tabs is an empty string ("").

Description

Use _tabs to define a series of tab stops. If _wrap is true, *dBASE Plus* ignores tab stops equal to or greater than _rmargin.

When you direct output to the printer, *dBASE Plus* maps each character according to the coordinate plane, a two-dimensional grid. The coordinate plane is divided into character cells

whose widths depend on the value of _ppitch. See the table in the description of _ppitch, which lists _ppitch values. The height of each character cell is determined by the size of the font.

If you change the value of _tabs, *dBASE Plus* takes the current value of _ppitch into consideration when calculating the cell width of the coordinate plane. This happens regardless of whether you're printing with proportional or monospaced fonts.

If you send a tab character, CHR(9), with ? or ??, *dBASE Plus* expands it to the amount of space required to reach the next tab stop. If the tab character you send is past the last tab stop, *dBASE Plus* ignores it, displaying output starting in the current column.

_wrap

Topic group Related topics Example

Determines if streaming output wraps between margins specified by _lmargin and _rmargin.

Syntax

_wrap = <expL>

<expL>

The logical expression *true* or *false*.

Default

The default for _wrap is *false*, which disables wrapping.

Description

Set _wrap to *true* to wrap streaming output from commands such as ?, ??, and LIST within the margins you specify with _lmargin and _rmargin.

When you enable wrapping, *dBASE Plus* wraps text onto the next line, breaking between words or numbers, when the output reaches the right margin. When you disable wrapping, *dBASE Plus* extends text beyond the right margin, moving to the next line only when a carriage return and linefeed combination (CR/LF) occurs in the text.

The print formatting commands _alignment, _indent, _lmargin, and _rmargin require _wrap to be *true.*

When _wrap is *true*, *dBASE Plus* stores streaming output in a buffer until it finishes displaying or printing the current line. If you generate output with the ? command, follow it with another ? command to force the last line of text to print.

ActiveX

class ActiveX

Topic group Related topics

Representation of an ActiveX control.

Syntax

[<oRef> =] new ActiveX(<container> [,<name expC>])

<oRef>

A variable or property—typically of <container>—in which to store a reference to the newly created ActiveX object.

<container>

The container—typically a Form object—to which you're binding the ActiveX object.

<name expC>

An optional name for the ActiveX object. If not specified, the ActiveX class will auto-generate a name for the object.

Properties

The following table lists the properties of interest in the ActiveX class. (No particular events or methods are associated with this class.)

Property	Default	Description
anchor	0 – None	How the ActiveX object is anchored in its container (0=None, 1=Bottom, 2=Top, 3=Left, 4=Right, 5=Center, 6=Container)
baseClassName	ACTIVEX	Identifies the object as an instance of the ActiveX class
classId		The ID string that identifies the ActiveX control
className	(ACTIVEX)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
description		A short description of the ActiveX control
nativeObject		The object that contains the ActiveX control's own properties, events, and methods

The following table lists the common properties, events, and methods of the ActiveX class:

Property		Event		Method
before	pageno	beforeRelease	onMouseMove	drag( )
borderStyle	parent	onClose	onRightDblClick	move( )
dragEffect	printable	onDragBegin	onRightMouseDown	release( )
form	speedTip	onLeftDblClick	onRightMouseUp	setFocus( )
height	systemTheme	onLeftMouseDown		
left	top	onLeftMouseUp		
name	width	onMiddleDblClick		
		onMiddleMouseDown		
		onMiddleMouseUp		

Description

An ActiveX object in *dBASE Plus* is a place holder for an ActiveX control, not an actual ActiveX control.

To include an ActiveX control in a form, create an ActiveX object on the form. Set the *classId* property to the component's ID string.Once the *classId* is set, the component inherits all the published properties, events, and methods of the ActiveX control, which are accessible through the *nativeObject* property. The object can be used just like a native *dBASE Plus* component.

New dBASE ActiveX Controls

dBASE has the ability to run ActiveX controls ever since they were introduced by Microsoft. Starting with dBASE PLUS 8, dBASE added additional ActiveX controls to the product enhance and augment the standard components that were included in previous dBASE releases.

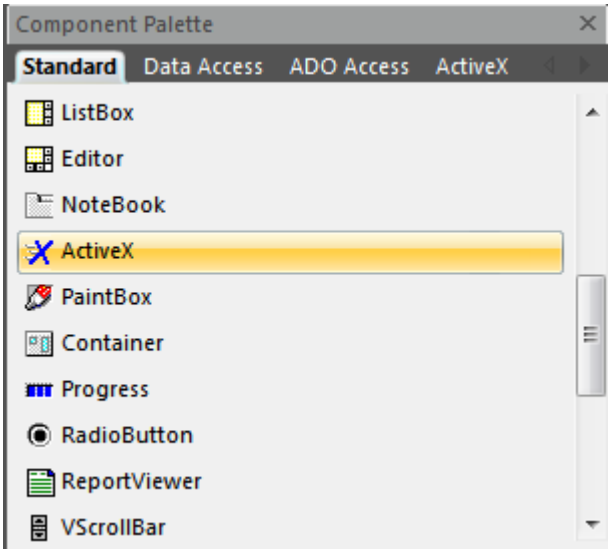
dBASE also supports 3rd party ActiveX components. The .ocx or .dll files that accompany ActiveX controls must be deployed with your application.

Note: *There are a few ActiveX controls that are not well-behaved and can cause issues with dBASE. One in particular is the Microsoft Silverlight OCX control. This control will make dBASE unstable and exit.* ***PLEASE DO NOT USE THAT CONTROL.***

Microsoft calls ActiveX objects, controls, or ActiveX controls. Once the ActiveX controls are added to dBASE's Component Palette they are referenced by either ActiveX controls or ActiveX components. In dBASE, the only difference is if the control is installed on the component palette, then it would be called a component.

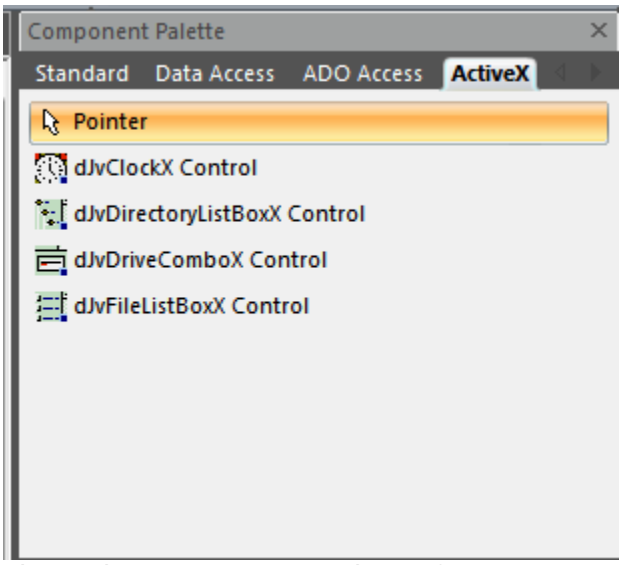
Installing new ActiveX controls in dBASE PLUS 9 or above

Installing ActiveX controls in dBASE is very easy and straightforward. There are two different methods for working with ActiveX controls in dBASE. The first is using an ActiveX container found on the main component palette as shown below:



Standard component palette – showing the ActiveX container

You just simply drop this component onto a form and set the **classID** under the Miscellaneous group in the Properties Editor. See the "Adding ActiveX components to the ActiveX component Palette" for more information. The second way to install the ActiveX controls on the ActiveX component palette.



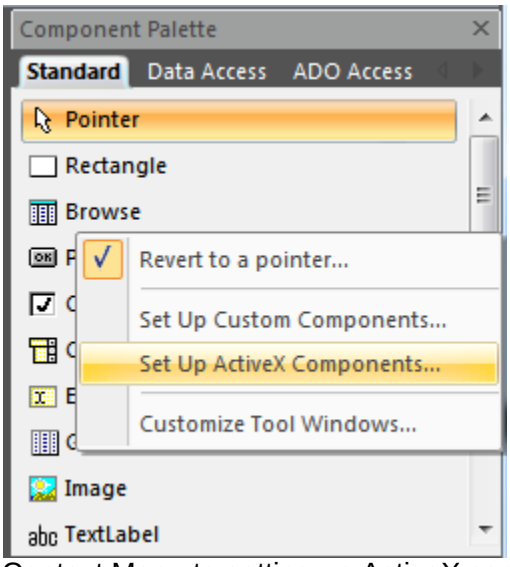
The ActiveX component palette after components are added

Adding components to the palette is very easy and simple and is covered in the next section.

Adding ActiveX components to the ActiveX component Palette

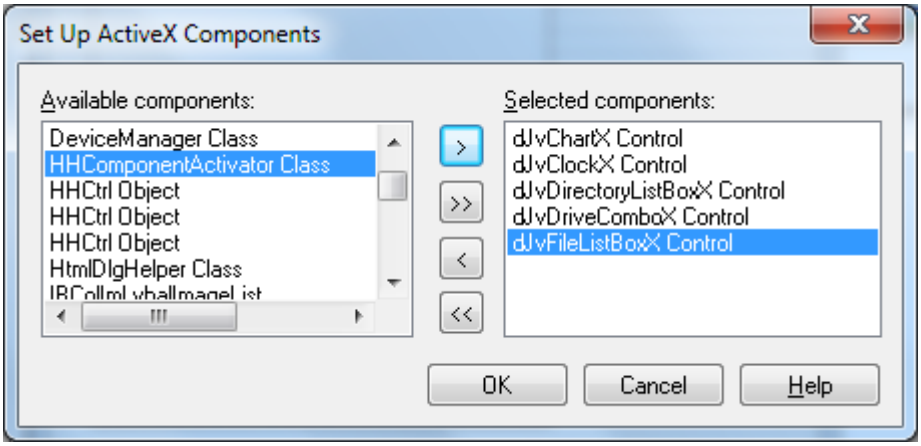
Once dBASE PLUS 9 or later has been installed, you can add the additional ActiveX controls to your component palette. dBASE already understands that the new components are ActiveX so that is not an issue.

1. Start dBASE
2. Create or Open a Form
3. Right mouse click on the component palette, the context menu should appear:



Context Menu to setting up ActiveX components

4. Select the Set Up ActiveX Components… menu item, this will display the following dialog:

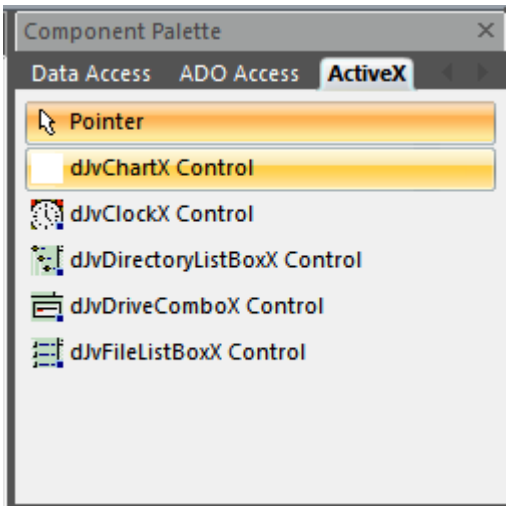


Pick the ActiveX components you want to have on the ActiveX component palette dialog

5. Use the >, >>, <, << buttons to select the ActiveX controls you want on the component palette and when done, press the OK button to finish.

a. Note: The above list of ActiveX Components are the new ones included in dBASE PLUS 8 and above.

6. Now notice the new tab on the Component Palette… it states ActiveX and it now has the 5 ActiveX components selected above.

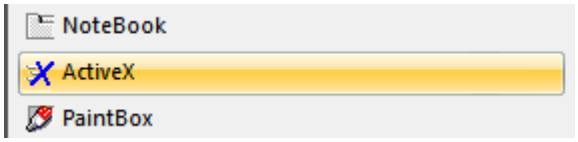


The added ActiveX components

7. Now the components are ready to use in your applications just like any other component.

Adding an ActiveX using the ActiveX container

On the main component page of dBASE you have an ActiveX container component.

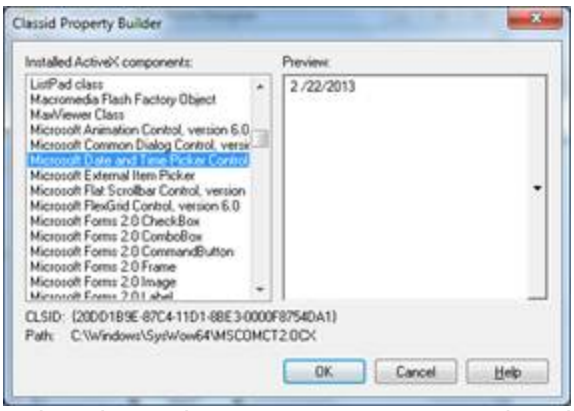


1. Choose that component and place it on the Form.

This is what the standard ActiveX container looks like before it has been assigned an ActiveX control.

Now in the Inspector for the ActiveX, you will notice a Miscellaneous area of properties. This is where you are going to focus, when setting up an ActiveX.

2. In the Inspector, set the classId property, click it, and select the Active wrench button in the property. This will start the Classid Property Builder dialog



Select the ActiveX you want to use and assign it to the ActiveX container

3. Most computers running Windows Vista and above will have the Microsoft Data and Time Picker Control as an ActiveX as shown selected above.

4. Click the OK button to pick that control to use

Microsoft's Date and Time Picker Control

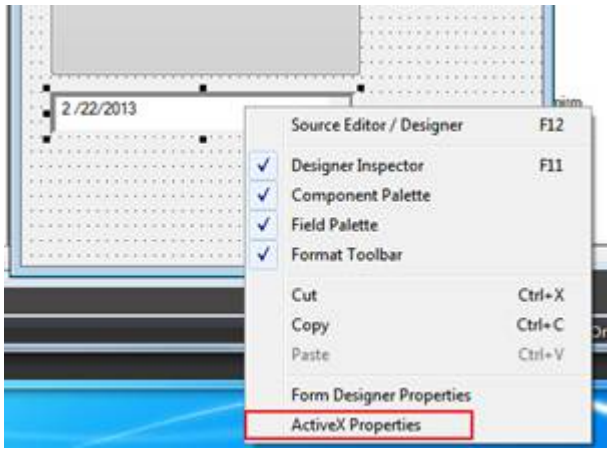
5. The ActiveX is now ready to use in dBASE.

Interacting with ActiveX(s)

Once you have picked an ActiveX to use, either by using the component palette or by using the ActiveX container the controls can be interacted with in the same way.

Properties Editors:

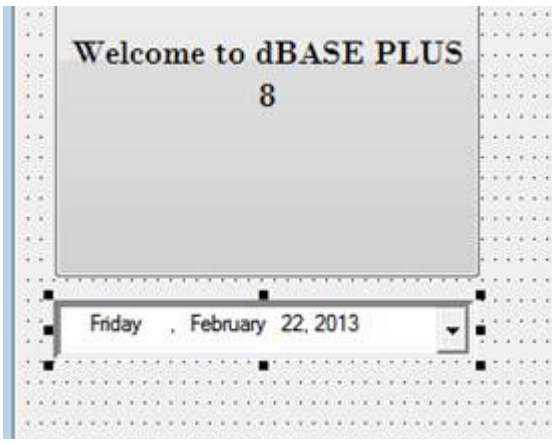
Some advanced ActiveX controls come with nice ActiveX property dialogs. Using the Microsoft Date and Time Picker Control – right-mouse click on the control, and the context menu will be shown:



Picking the ActiveX Properties from the context menu

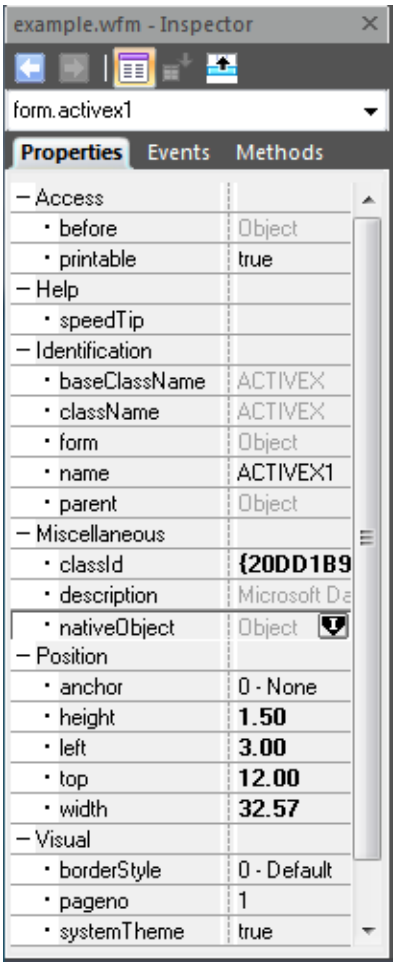
Selectable options for the ActiveX

Notice changing a property in the ActiveX dialog will automatically show in the dBASE designer being used



The changes in the dBASE designer after clicking the Apply button on the ActiveX properties dialog.

The other way to interact with ActiveX components is to use the Inspector and the nativeObject property. This method will use the ActiveX's Properties, Events, and Methods to control the component.



Notice the nativeObject has the Inspect icon ready to click.

Notice the Native properties for the Control. These properties can be set as any other property either in the Inspector or by code.

Clicking the Events tab up at the top will give the ActiveX's registered events.

Clicking the Methods tab in the Inspector will give you the registered methods for the ActiveX control.

This works for both ActiveX
controls and ActiveX containers

Auto propagation of ActiveX components

After using the ActiveX container in dBASE, you may notice that the ActiveX is now on the ActiveX component palette. This is due to the nature of the ActiveX control. If it is a well-behaved control, it will self publish to the ActiveX component palette found inside dBASE.

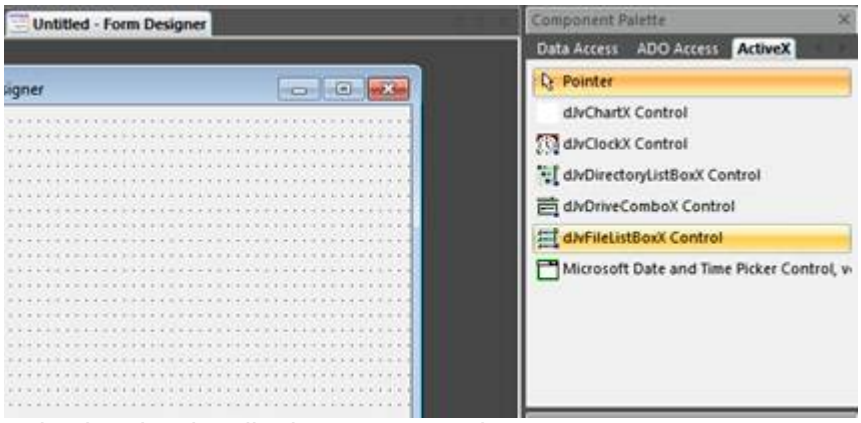
Note: *Remember that not all components are well behaved and some can make dBASE unstable or even crash. There are 1000's of ActiveX(s) on the market today and internal and external testing of each component will be needed to ensure proper functionality.*

Using an ActiveX container and Control in the same project

This simple example will provide a general overview of using ActiveX controls in an application.

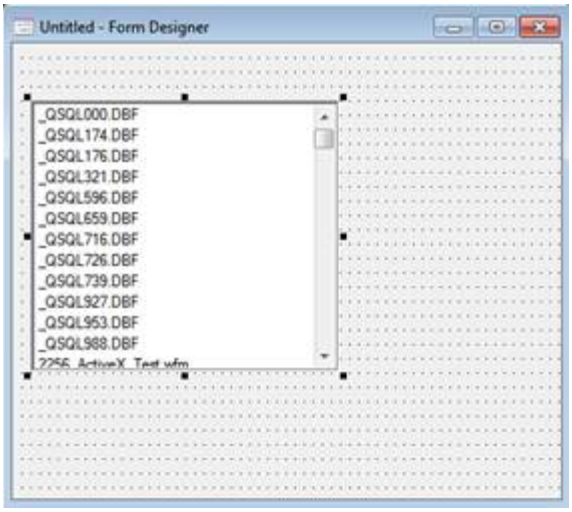
1. Start dbASE
2. Create a new form
3. Select ActiveX tab

. Note: this example makes the assumption that you have already added the ActiveX controls to the Component palette.
You will find the steps here: Adding ActiveX to dBASE PLUS 9



Selecting the dJvFileListBoxX control

4. Click the dJvFileListBoxX control and add it to the form



Form after ActiveX component is dropped on it, showing live data

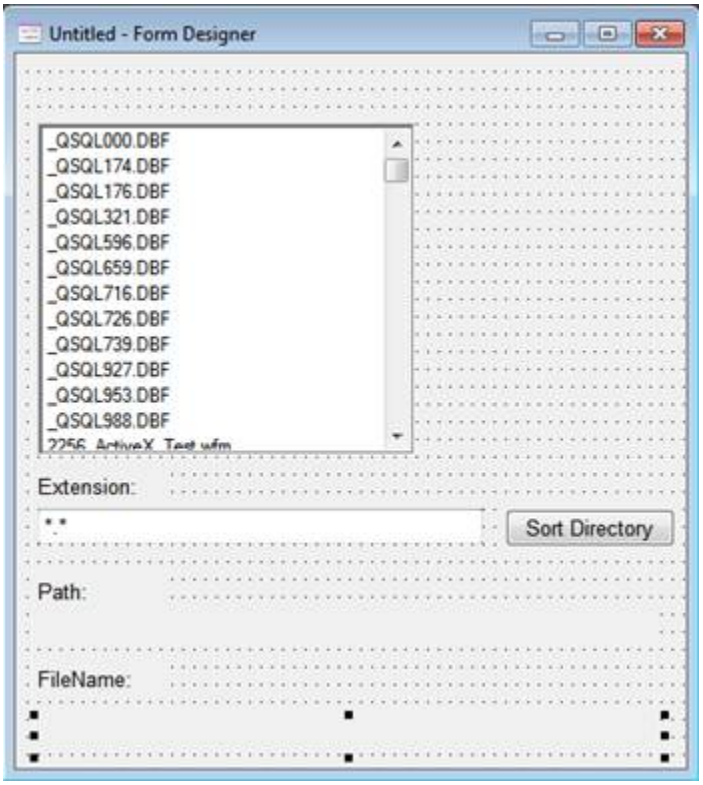
5. Now add a few components:

a. 5 – Textlables – these are for information

b. 1 – Entryfield – this is for the extensions

c. 1 – Pushbutton – this is for sorting on the extension

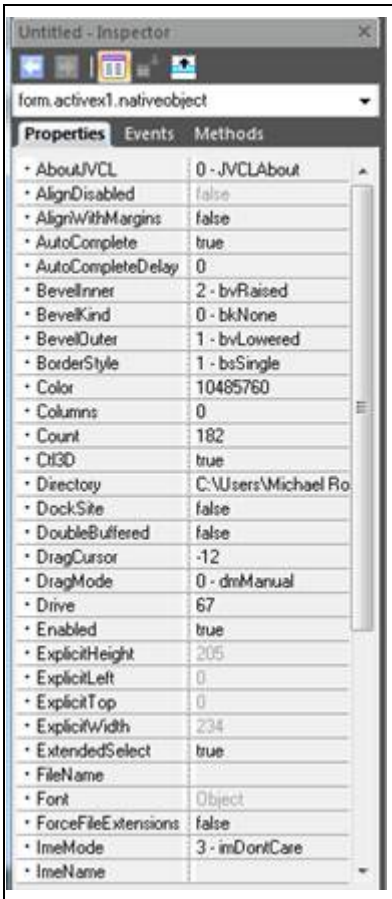
The form layout should look something like this:



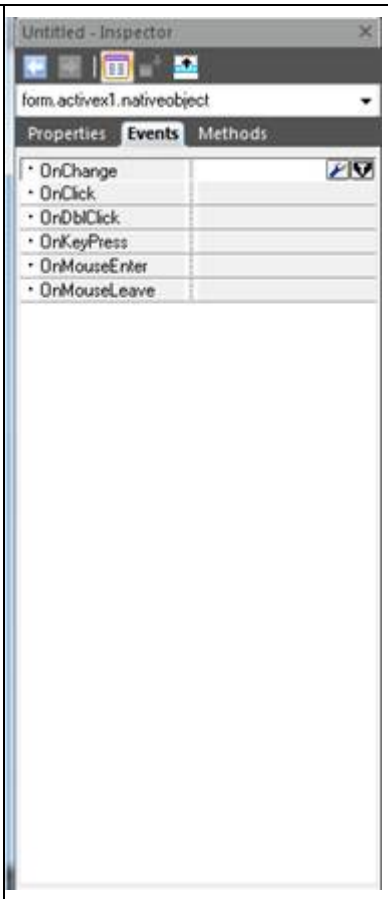
After the objects have been setup and labeled

6. Now the components need to be wired to work with the interface.

7. The first step is to connect the file list to the Path and FileName labels. This can be done with the ActiveX – Native Properties, Events, and Methods. To get to the nativeObject properties, click the nativeObject property of the control, to display the following :



nativeObject's Properties



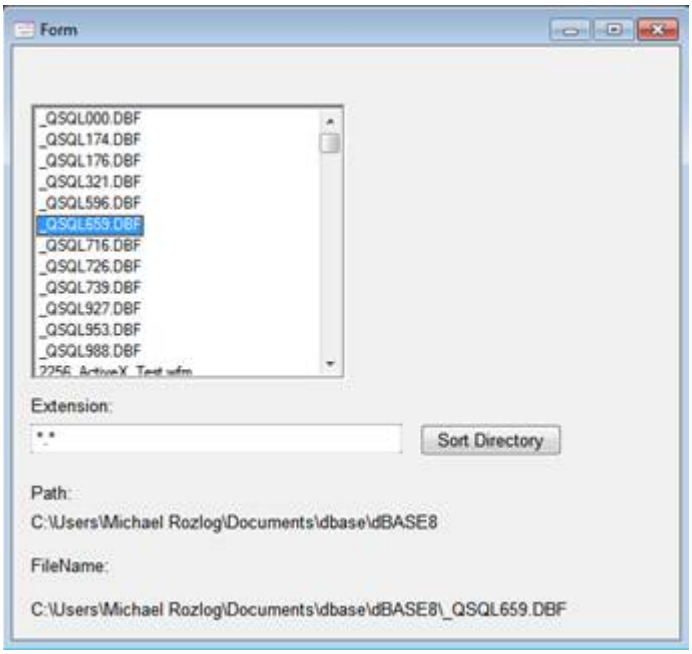
nativeObject's Events

nativeObject's Methods

8. We can use the nativeObject's Events – OnChange to update the Textlabel objects on our form. Click on the Events tab for the nativeObject, then click the active wrench on the OnChange event.

```
function nativeObject_OnChange
form.textlabel3.text = form.activex1.nativeobject.directory
form.textlabel5.text = form.activex1.nativeobject.filename
return
```

9. Now, Save the form and test it out. You should notice that every time you change the file selected, the FileName: text changes.



Application running and showing the Path and FileName

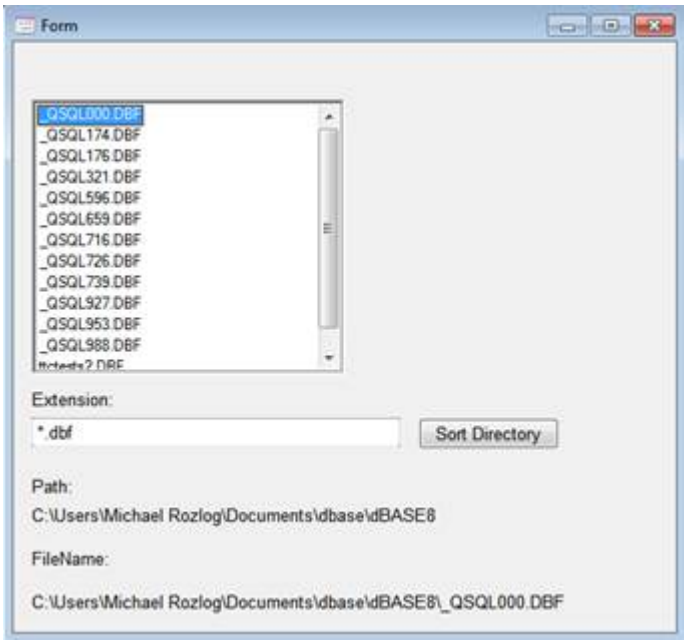
10. Back to Design mode as we have a few more things so add:

In this example, the directory that is being used has various dBASE file types. One of the most common is the *.dbf. The next steps will be to show how to filter just those files in the interface.

11. Now click the PushButtons's OnClick Event and add the following code

```
function PUSHBUTTON1_onClick
  form.activex1.nativeobject.mask = form.entryfield1.value
  return
```

12. Save and run the Form. To display only the .dbf files in the example directory, type "*.dbf" in the extension Entryfield and press the Sort Directory button. Notice the FileListBox will only show the files with those extensions.



Application showing only *.dbf files in the dJvFileListBoxX

13. The final task we want to show with our dJvFileListBoxX ActiveX component is to add the graphic glyphs back to the files listed for easier recognition.

14. Add another PushButton

a. Name it Show Glyphs

15. Click the new PushButton's|Events|OnClick and add the following code:

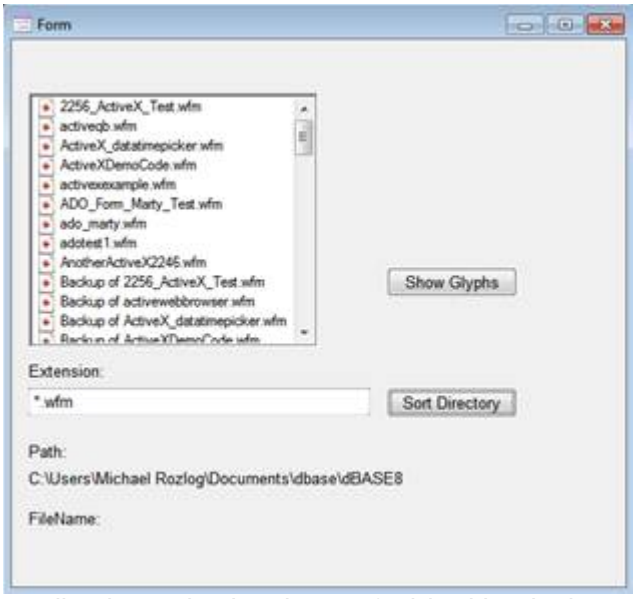
```
Function PUSHBUTTON2_onClick
form.activex1.nativeobject.showglyphs = true
return
```

16. Now Save, and Execute the form:

a. Click the Show Glyphs button

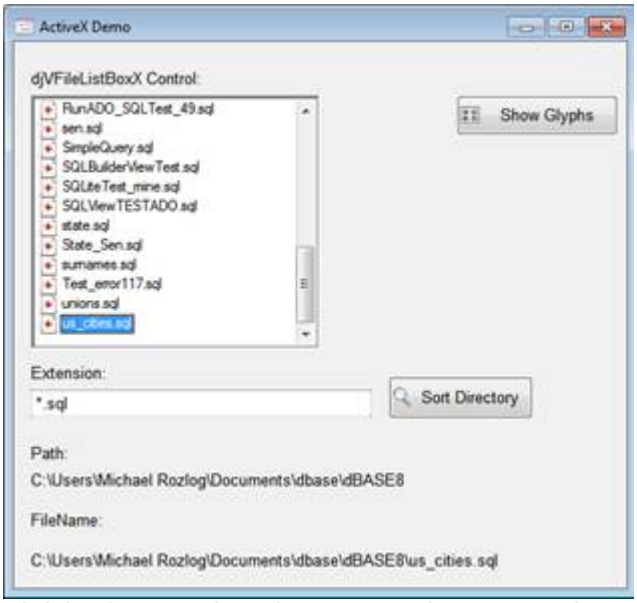
b. Put in an Extension – for this example I will use *.wfm

c. Results should look something like:



Application only showing *.wfm(s) with Glyphs turned on

17. In this example, we learned how to mix native dBASE forms and controls with the new ActiveX controls. There are many other things that can be done using these new controls and this is just a simple introduction. The sky is the limit to what you can accomplish with the new controls.



Finished Example using new ActiveX controls

djvClockX

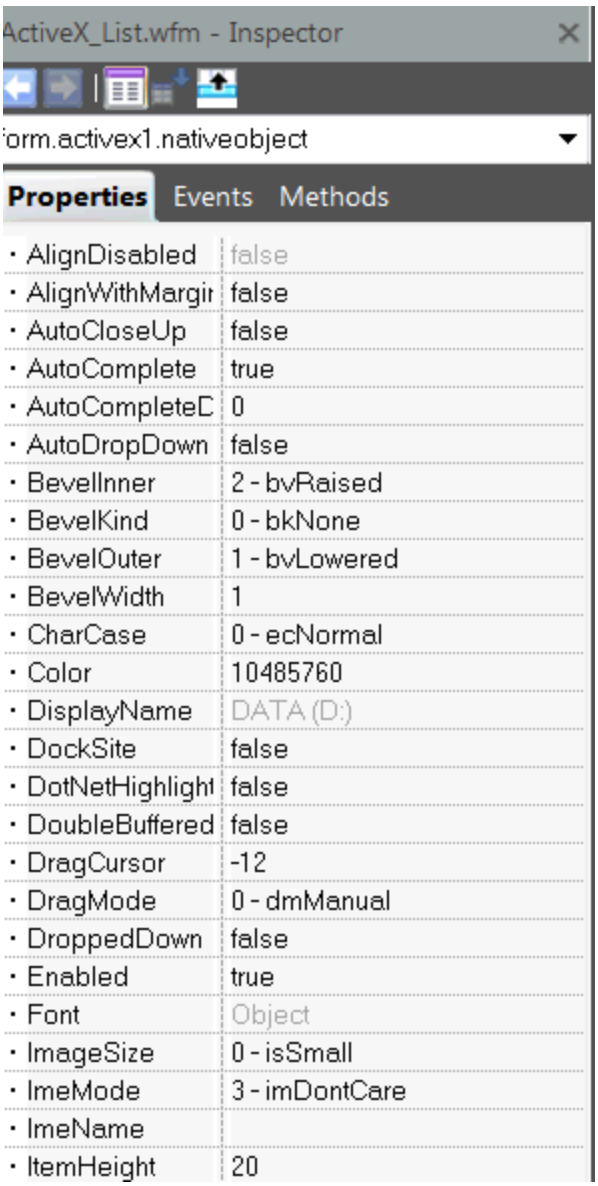
Topic group

The dJvClockX ActiveX is used displaying time and dates on the screen and also comes with advanced features like Alarms .

The following example includes an ActiveX component on a Form... (NOTE: See 'Using an ActiveX container and Control...')

```
class activeXForm of FORM
   with (this)
      height = 16.0
      left = 86.5
      top = 0.0
      width = 40.0
      text = ""
   endwith
   this.ACTIVEX1 = new ACTIVEX(this)
   with (this.ACTIVEX1)
      height = 4.0
      left = 14.0
      top = 6.16
      width = 12.0
      classId = "{8627E73B-B5AA-4643-A3B0-570EDA17E3E7}" //derived from registry
   endwith
```

Any Properties, Events or Methods can be found by viewing the 'nativeObject' property for the ActiveX component in the navigator...



djvDirectoryListBox

Topic group

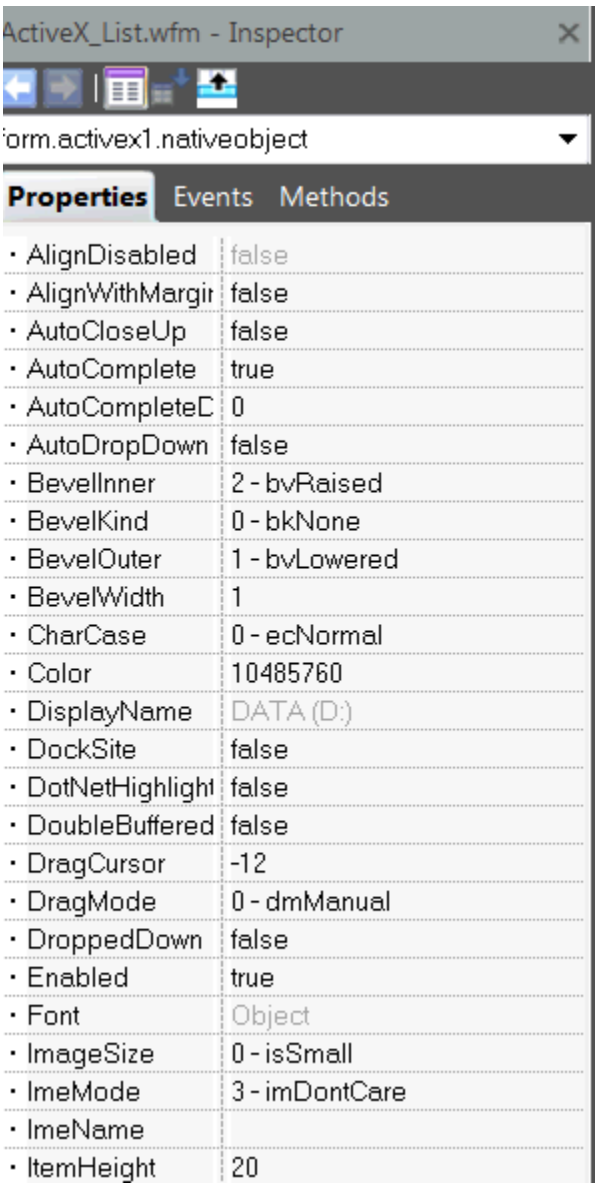
The dJvDirectoryListBoxX ActiveX is used displaying a list of directories on a registered drive. .

The following example includes an ActiveX component on a Form... (NOTE: See 'Using an ActiveX container and Control...')

```
class activeXForm of FORM
   with (this)
      height = 16.0
      left = 86.5
```

```
   top = 0.0
   width = 40.0
   text = ""
endwith
this.ACTIVEX1 = new ACTIVEX(this)
with (this.ACTIVEX1)
   height = 4.0
   left = 14.0
   top = 6.16
   width = 12.0
   classId = "{8627E73B-B5AA-4643-A3B0-570EDA17E3E7}" //derived from registry
endwith
```

Any Properties, Events or Methods can be found by viewing the 'nativeObject' property for the ActiveX component in the navigator...



djvDriveComboX

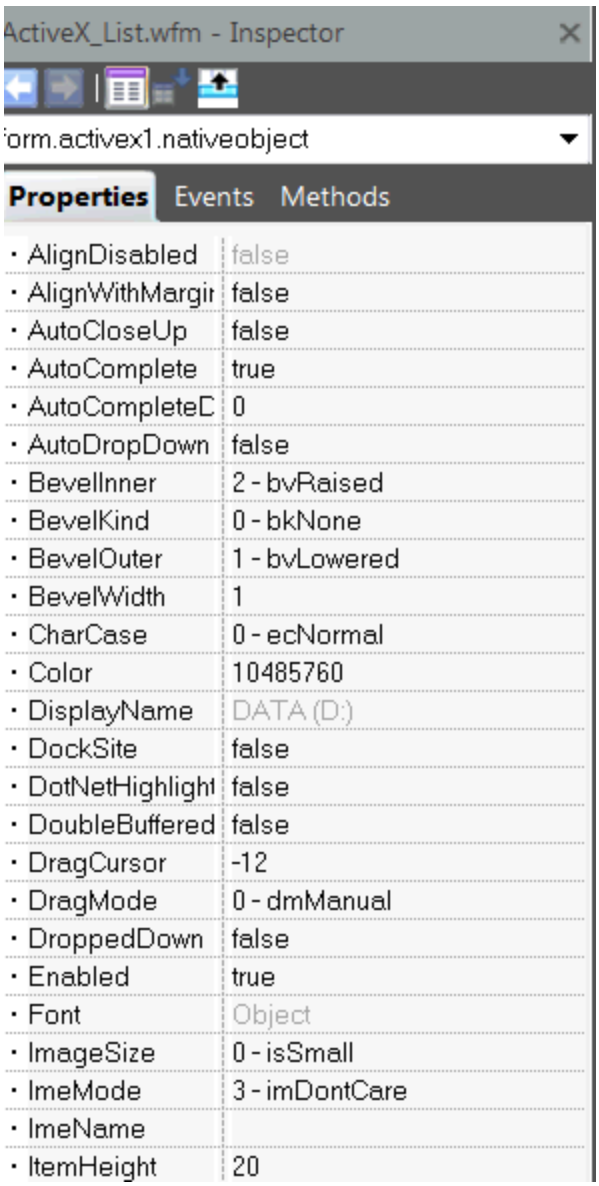
Topic group

The dJvDriveComboX ActiveX is used for displaying the registered drives in a dropdown combobox

The following example includes an ActiveX component on a Form... (NOTE: See 'Using an ActiveX container and Control...')

```
class activeXForm of FORM
   with (this)
      height = 16.0
      left = 86.5
      top = 0.0
      width = 40.0
      text = ""
   endwith
   this.ACTIVEX1 = new ACTIVEX(this)
   with (this.ACTIVEX1)
      height = 4.0
      left = 14.0
      top = 6.16
      width = 12.0
      classId = "{8627E73B-B5AA-4643-A3B0-570EDA17E3E7}" //derived from registry
   endwith
```

Any Properties, Events or Methods can be found by viewing the 'nativeObject' property for the ActiveX component in the navigator...



djvFileListBoxX

Topic group

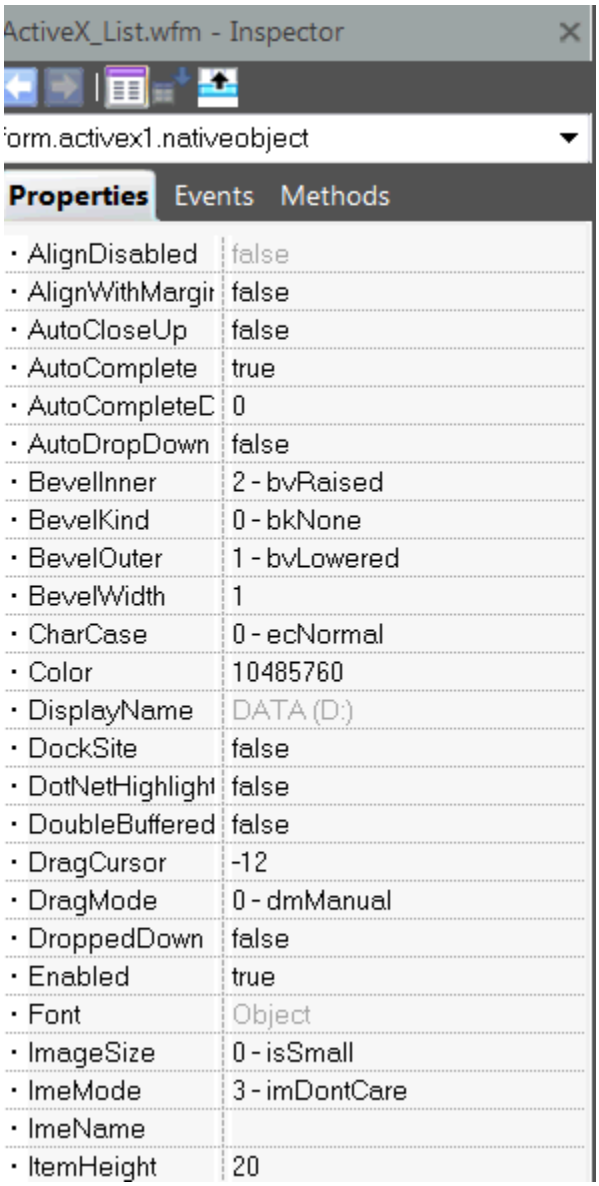
The dJvFileListBoxX ActiveX is used for displaying files in a directory

The following example includes an ActiveX component on a Form... (NOTE: See 'Using an ActiveX container and Control...')

```
class activeXForm of FORM
```

```
with (this)
   height = 16.0
   left = 86.5
   top = 0.0
   width = 40.0
   text = ""
endwith
this.ACTIVEX1 = new ACTIVEX(this)
with (this.ACTIVEX1)
   height = 4.0
   left = 14.0
   top = 6.16
   width = 12.0
   classId = "{8627E73B-B5AA-4643-A3B0-570EDA17E3E7}" //derived from registry
endwith
```

Any Properties, Events or Methods can be found by viewing the 'nativeObject' property for the ActiveX component in the navigator...



djvRichEditX

Topic group

The dJvRichEditX ActiveX is used for creating rich text edits with Printing, Undo, Redo and a whole lot more

The following example includes an ActiveX component on a Form... (NOTE: See 'Using an ActiveX container and Control...')

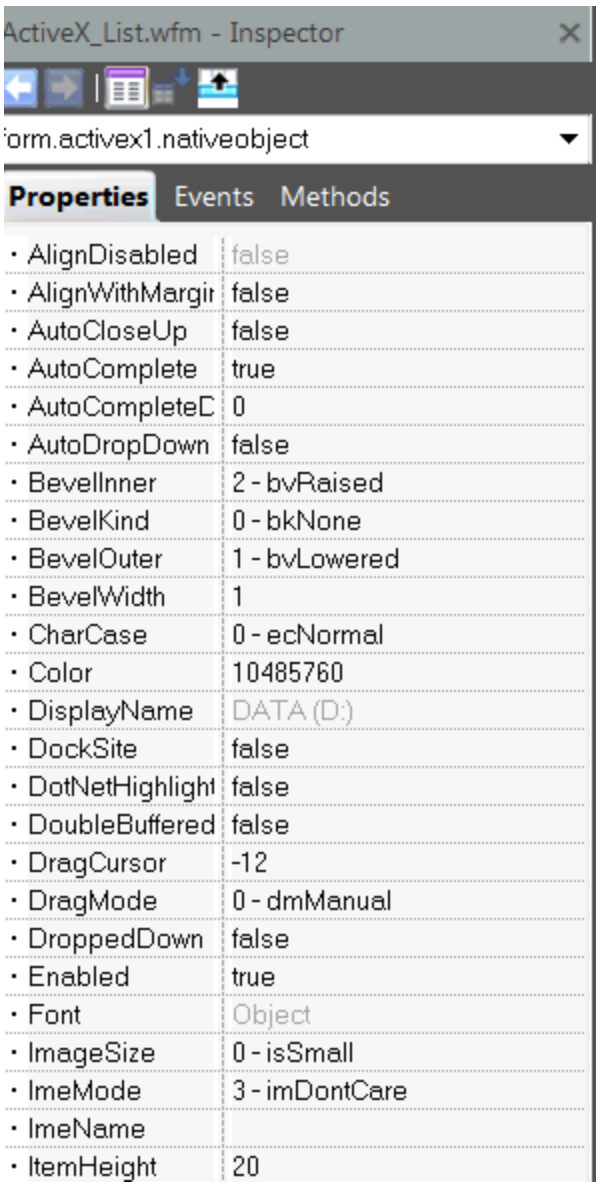
```
class activeXForm of FORM
   with (this)
      height = 16.0
      left = 86.5
      top = 0.0
      width = 40.0
      text = ""
   endwith
   this.richedit1 = new ACTIVEX(this)
   with (this.ACTIVEX1)
      height = 4.0
      left = 14.0
      top = 6.16
      width = 12.0
      classId = "{8627E73B-B5AA-4643-A3B0-570EDA17E3E7}" //derived from registry
   endwith
```

Example:

Selecting some text from the component:

```
form.richedit1.nativeobject.selstart = 0
form.richedit1.nativeobject.sellength = 5
form.text1.text = form.richedit1.nativeobject.seltext
```

Any Properties, Events or Methods can be found by viewing the 'nativeObject' property for the ActiveX component in the navigator...



Bitwise

Bitwise

Topic group

The functions in this chapter are used for bit manipulation and base conversion for unsigned 32-bit values. These values are often passed to and returned by Windows API and other DLL functions. Interpreting such values often requires analysis and manipulation of individual bits.

For all parameters designated as 32-bit integers, non-integers will be truncated to integers. For integers larger than 32 bits, only the least significant (right-most) 32 bits are used.

BITAND( )

Topic group Related topics Example

Performs a bitwise AND.

Syntax

BITAND(<expN1>, <expN2>)

<expN1>
<expN2>

Two 32-bit integers

Description

BITAND( ) compares bits in the numeric value <expN1> with corresponding bits in the numeric value <expN2>. When both bits in the same position are on (set to 1), the corresponding bit in the returned value is on. In any other case, the bit is off (set to 0).

AND	0	1
0	0	0
1	0	0

Use BITAND( ) to force individual bits to zero. Create a bit mask: a 32-bit integer with zeroes in the bits you want to force to zero and ones in the bits you want to leave alone. Use this bit mask as either one of the parameters to BITAND( ), and the other parameter as the number that is modified.

BITLSHIFT( )

Topic group Related topics Example

Shifts a number's bits to the left.

Syntax

BITLSHIFT(<int expN>, <shift expN>)

<int expN>

A 32-bit integer.

<shift expN>

The number of places to shift, from 0 to 32.

Description

BITLSHIFT( ) moves each bit in the numeric value <int expN> to the left the number of times you specify in <shift expN>. Each time the bits are shifted, the least significant bit (bit 0, the bit farthest to the right) is set to 0, and the most significant bit (bit 31, the bit farthest to the left) is lost.

Shifting a number’s bits to the left once has the effect of multiplying the number by two, except that if the number gets too large—equal to or greater than 2^32 (roughly 4 billion)—the high bit is lost.

BITNOT( )

Topic group Related topics

Inverts the bits in a number

Syntax

BITNOT(<expN>)

<expN>

A 32-bit integer.

Description

BITNOT( ) inverts all 32 bits in <expN>. All zeroes become ones, and all ones become zeroes.

To invert specific bits, use BITXOR( ).

BITOR( )

Topic group Related topics

Performs a bitwise OR.

Syntax

BITOR(<expN1>, <expN2>)

<expN1>
<expN2>

Two 32-bit integers

Description

BITOR( ) compares bits in the numeric value <expN1> with corresponding bits in the numeric value <expN2>. When either or both bits in the same position are on (set to 1), the corresponding bit in the returned value is on. When neither element is on, the bit is off (set to 0).

OR	0	1
0	0	1
1	1	1

Use BITOR( ) to force individual bits to one. Create a bit mask: a 32-bit integer with ones in the bits you want to force to one and zeroes in the bits you want to leave alone. Use this bit mask as either one of the parameters to BITOR( ), and the other parameter as the number that is modified.

BITRSHIFT( )

Topic group Related topics

Shifts a number's bits to the right, maintaining sign.

Syntax

BITRSHIFT(<int expN>, <shift expN>)

<int expN>

A signed 32-bit integer.

<shift expN>

The number of places to shift, from 0 to 32.

Description

Unlike the other bitwise functions, BITRSHIFT( ) treats its 32-bit integer as a signed 32-bit integer. The sign of a 32-bit integer is stored in the most significant bit (bit 31), which is also referred to as the high bit. If the high bit is 1, the number is negative if it is treated as a signed integer. Otherwise, it is simply a very large unsigned integer.

BITRSHIFT( ) moves each bit in the numeric value <int expN> to the right the number of times you specify in <shift expN>. Each time the bits are shifted, the previous value of the high bit is restored, and the least significant bit (bit 0, the bit farthest to the right) is lost. This is called a sign-extended shift, because the sign is maintained.

A similar function, BITZRSHIFT( ), performs a zero-fill right shift, which always sets the high bit to zero. If <int expN> is a positive integer less than 2^31, BITZRSHIFT( ) and BITRSHIFT( ) have the same effect, because the high bit for such an integer is zero.

Use BITRSHIFT( ) when you're treating the integer as a signed integer. Use BITZRSHIFT( ) when the integer is unsigned.

Shifting a number's bits to the right once has the effect of dividing the number by two, dropping any fractions.

BITSET( )

Topic group Related topics Example

Checks if a specified bit in a numeric value is on.

Syntax

BITSET(<int expN>, <bit expN>)

<int expN>

A 32-bit integer.

<bit expN>

The bit number, from 0 (the least significant bit) to 31 (the most significant bit).

Description

BITSET( ) evaluates the number <int expN> and returns *true* if the bit in position <bit expN> is on (set to 1), or *false* if it is off (set to 0). For example, the binary representation of 3 is

00000000 00000000 00000000 00000011

bit number 0 is on, bit number 2 is off.

BITXOR( )

Topic group Related topics

Performs a bitwise exclusive OR.

Syntax

BITXOR(<expN1>, <expN2>)

<expN1>
<expN2>

Two 32-bit integers

Description

BITXOR( ) compares bits in a numeric value <expN1> with corresponding bits in the numeric value <expN2>. When one (and only one) of two bits in the same position are on (set to 1), the corresponding bit in the returned value is on. In any other case, the bit is off (set to 0).

XOR	0	1
0	0	1
1	1	0

This operation is known as exclusive OR, since one bit (and only one bit) must be set on for the corresponding bit in the returned value to be set on.

Use BITXOR( ) to flip individual bits. Create a bit mask: a 32-bit integer with ones in the bits you want to flip and zeroes in the bits you want to leave alone. Use this bit mask as either one of the parameters to BITXOR( ), and the other parameter as the number that is modified.

BITZRSHIFT( )

Topic group Related topics Example

Shifts a number's bits to the right.

Syntax

BITZRSHIFT(<int expN>, <shift expN>)

<int expN>

A 32-bit integer.

<shift expN>

The number of places to shift, from 0 to 32.

Description

BITZRSHIFT( ) moves each bit in the numeric value <int expN> to the right the number of times you specify in <shift expN>. Each time the bits are shifted, the most significant bit (bit 31, the bit farthest to the left) is set to 0, and the least significant bit (bit 0, the bit farthest to the right) is lost.

Shifting a number’s bits to the right once has the effect of dividing the number by two, dropping any fractions.

Like most other bitwise functions, BITZRSHIFT( ) treats <int expN> as an unsigned integer. To shift a signed integer, use BITRSHIFT( ) instead.

HTOI( )

Topic group Related topics Example

Returns the numeric value of a specified hexadecimal number.

Syntax

HTOI(<expC>)

<expC>

The hexadecimal number whose numeric value to return.

Description

Use HTOI( ) to convert a string containing a hexadecimal number to its numeric value (in decimal). For example, you might allow the input of a hexadecimal number. This input would have to go into a string because the hexadecimal digits A through F are considered characters. To use the hexadecimal number, you would have to convert the hexadecimal string into its numeric value.

HTOI( ) will attempt to convert a hexadecimal number of any magnitude; it is not limited to 32 bits (8 hexadecimal digits).

You may specify literal hexadecimal numbers by preceding them with 0x; HTOI( ) is not necessary. For example, 0x64 and HTOI("64") result in the same number: 100 decimal.

ITOH( )

Topic group Related topics

Returns the hexadecimal equivalent of a specified number, as a character string.

Syntax

ITOH(<int expN> [, <chars expN>])

<int expN>

The 32-bit integer whose hexadecimal equivalent to return.

<chars expN>

The minimum number of characters to include in the returned hexadecimal character string.

Description

Use ITOH( ) to convert a number to a character string representing its hexadecimal equivalent. The hexadecimal number may be used for display and editing/input purposes. To use the hexadecimal number as a number, it must be converted back into a numeric value with HTOI( ).

By default, ITOH( ) uses only as many characters as necessary to represent <int expN> in hexadecimal. If <chars expN> is greater than the number of characters required, ITOH( ) pads the returned string with leading 0's to make it <chars expN> characters long. If <chars expN> is less than the number of characters required, it is ignored. For example, ITOH(21) returns the string "15", while ITOH(21,4) returns "0015".

Because ITOH( ) treats the integer as a 32-bit integer, negative integers are always converted into 8 hexadecimal digits. For example, ITOH(-1) returns "FFFFFFFF".

Miscellaneous Language Elements

Everything else (except Preprocessor)

Topic group

This section includes dBL language elements that pertain to errors, security, and locale.

ACCESS( )

Topic group Related topics

Returns the access level of the current user for DBF table security.

Syntax

ACCESS( )

Description

In DBF table security, an access level is assigned to each user. The access level is a number from 1 to 8, with 1 being the highest level of access. Use ACCESS( ) to build security into an application. The access level returned can be used to test privileges assigned with PROTECT. If a user is not logged in to the application, ACCESS( ) returns 0 (zero).

If you write programs that use encrypted files, check the user's access level early in the program. If ACCESS( ) returns zero, your program might prompt the user to log in, or to contact the system administrator for assistance.

For more information, see PROTECT.

ANSI( )

Topic group Related topics Example

Returns a character string that is the ANSI equivalent of a character expression using the current global character set.

Syntax

ANSI(<expC>)

<expC>

The character expression to convert to ANSI characters.

Description

Each character in a string is represented by a byte value from 0 to 255. The character each number represents is determined by the current character set. The same series of bytes may represent different characters with different character sets. Conversely, the same character may be represented by a different byte value in different character sets.

Windows uses the ANSI (American National Standards Institute) character set. *dBASE Plus* supports that character set, and multiple OEM (Original Equipment Manufacturer) character sets, which are identified by a code page number. (For more information, see About character sets.) Each character set, along with other country-specific information, is represented in *dBASE Plus* by a language driver. The classic IBM extended character set—the one with box drawing characters used in text screens and the MS-DOS Command Prompt—is an OEM character set, represented by the DB437US0 language driver, the default language driver for the United States.

There are more characters in use than will fit in a 256-character character set; therefore some characters are present in some character sets but not in others. While the lower 128 characters of these character sets are always identical (they match the standard 7-bit ASCII characters), the upper 128 characters (sometimes referred to as high-ASCII characters) may differ. Sometimes the same characters have different byte values. For example, the lowercase a-umlaut (ä) is character 132 in the DB437US0 OEM character set, and character 228 in the ANSI character set.

Use the ANSI( ) and OEM( ) functions to convert characters between the ANSI character set and an OEM character set, represented by the current global language driver.

Note

If the current language driver is an ANSI language driver, like DBWINUS0, then DB437US0 is used as the OEM character set.

ANSI( ) treats the byte values of the characters in <expC> as OEM characters, and attempts to convert them to the equivalent characters in the ANSI character set. OEM( ) does the reverse. If no direct conversion is possible, then the characters are converted to similar-looking characters.

CANCEL

Topic group Related topics

Halts program execution.

Syntax

CANCEL

Description

Use CANCEL to cancel program execution in the midst of a process. You can also issue CANCEL in the Command window when a program is suspended (with SUSPEND) to cancel execution of the suspended program.

While procedural applications are characterized by deeply nested subroutines that wait for user actions, applications in *dBASE Plus* are event-driven; objects sit on-screen, waiting for something to happen. While waiting for an event, no programs are being executed. When an event occurs, the event handler is fired, and when that's done, *dBASE Plus* goes back to waiting for events. In the dBASE IDE, issuing CANCEL will halt the current event handler thread, but does not cause dBASE Plus to stop responding to events (similar to choosing the Cancel option from a dBASE error dialog) . To cause dBASE objects to stop responding to events the object itself must be removed from the screen or destroyed.

A process that is halted simply stops; no message or exception is generated. In the main routine of a process, issuing CANCEL has the same effect as issuing RETURN: the process is terminated. A program or thread halted by CANCEL performs the standard cleanup for a completed process. All local and private memory variables are cleared. Control returns to the object that started the process, if it's still available; usually a form, menu, or the Command window.

In the dBASE Runtime, issuing CANCEL will halt execution of an application and cause it to shutdown.

CERROR( )

Topic group Related topics Example

Returns the number of the last compiler error.

Syntax

CERROR( )

Description

Use CERROR( ) before executing a new program to test whether the source code compiles successfully. If no compiler error occurs, CERROR( ) returns 0. CERROR( ) is updated each

time you or *dBASE Plus* compiles a program or format file. CERROR( ) isn't affected by warning messages generated by compiling.

Use CERROR( ) in a program file. If you issue ? CERROR( ) in the Command window, it returns 0. (This is because dBASE is compiling the "? CERROR( )" command itself, which does not cause a compiler error.)

See the table in the description of ERROR( ) that compares ERROR( ), MESSAGE( ), DBERROR( ), DBMESSAGE( ), SQLERROR( ), SQLMESSAGE( ), and CERROR( ).

CHARSET( )

Topic group Related topics Example

Returns the name of the character set the current table or a specified table is using. If no table is open and you issue CHARSET( ) without an argument, it returns the global character set in use.

Syntax

CHARSET([<alias>])

<alias>

A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

Use CHARSET( ) to learn which character set the current table or a specified table is using. If you don't pass CHARSET( ) an argument, it returns the name of the character set of the current table or, if no tables are open, the global character set in use. CHARSET( ) also returns information on Paradox and SQL databases.

The character set in which a table's data is stored depends on the language driver setting that was in effect when the table was created. With *dBASE Plus*, you can choose the language driver that applies to your *dBASE Plus* data in the [CommandSettings] section in the PLUS.ini file.

The value CHRSET( ) returns is a subset of the value LDRIVER( ) returns. For more information, see LDRIVER( ).

DBASE_SUPPRESS_STARTUP_DIALOGS

An operating system environment variable which allows suppression of runtime engine initialization error dialogs.

Syntax

DBASE_SUPPRESS_STARTUP_DIALOGS=1 will turn on suppression of the startup dialogs

DBASE_SUPPRESS_STARTUP_DIALOGS= will turn off suppression of the startup dialogs

Description

The purpose of the DBASE_SUPPRESS_STARTUP_DIALOGS variable is to prevent the display of any error dialogs during the startup of the *dBASE Plus* runtime engine (PLUSrun.exe). The suppressed error dialogs are those which can occur before any application code is executed by the runtime and as a result, cannot be trapped via a try...catch command.

The potential problem arises due to the fact that once an application is launched, users will not see these dialogs and, therefore, will not be aware of being prompted for a response. When suppression of an error dialog occurs, *dBASE Plus* will shut itself down, thereby preventing instances of the runtime engine from becoming stranded in memory.

Setting dBASE Plus dialog suppression

Windows 95/98 - Set DBASE_SUPPRESS_STARTUP_DIALOGS in the autoexec.bat file OR the .bat file used to launch the *dBASE Plus* web application .exe.

Windows 2000 or NT - Set DBASE_SUPPRESS_STARTUP_DIALOGS from the Environment tab of the System Properties dialog - which can be found in the Control Panel window.

DBERROR( )

Topic group Related topics Example

Returns the number of the last BDE error.

Syntax

DBERROR( )

Description

DBERROR( ) returns the BDE error number of the last BDE error generated by the current table. To learn the BDE error message itself, use DBMESSAGE( ).

See the table in the description of ERROR( ) that compares ERROR( ), MESSAGE( ), DBERROR( ), DBMESSAGE( ), SQLERROR( ), SQLMESSAGE( ), and CERROR( ).

DBMESSAGE( )

Topic group Related topics Example

Returns the error message of the last BDE error.

Syntax

DBMESSAGE( )

Description

DBMESSAGE( ) returns the error message of the most recent BDE error.

See the table in the description of ERROR( ) that compares ERROR( ), MESSAGE( ), DBERROR( ), DBMESSAGE( ), CERROR( ), SQLERROR( ), and SQLMESSAGE( ).

ERROR( )

Topic group Related topics Example

Returns the number of the most recent *dBASE Plus* error.

Syntax

ERROR( )

Description

Use ERROR( ) to determine the error number when an error occurs. ERROR( ) is initially set to 0. ERROR( ) returns an error number when an error occurs, and remains set to that number until one of the following happens:

Another error occurs
RETRY is issued
The subroutine in which the error occurs completes execution

The following table compares the functionality of CERROR( ), DBERROR( ), DBMESSAGE( ), ERROR( ), MESSAGE( ), SQLERROR( ), and SQLMESSAGE( ).

Function	Returns
CERROR( )	Compiler error number
DBERROR( )	BDE error number
DBMESSAGE( )	BDE error message
ERROR( )	dBASE error number
MESSAGE( )	dBASE error message
SQLERROR( )	Server error number
SQLMESSAGE( )	Server error message

ID( )

Topic group Related topics Example

Returns the name of the current user on a local area network (LAN) or other multiuser system.

Syntax

ID( )

Description

ID( ) accepts no arguments and returns the name of the current user as a character string. ID( ) returns an empty string when you call it on a single-user system or when a user name isn't registered on a multiuser system.

LDRIVER( )

Topic group Related topics Example

Returns the name of the language driver the current table or a specified table is using. If no table is open and you issue LDRIVER( ) without an argument, it returns the global language driver in use.

Syntax

LDRIVER([<alias>])

<alias>

A work area number (1 through 225), letter (A through J), or alias name. The work area letter or alias name must be enclosed in quotes.

Description

Use LDRIVER( ) to learn which language driver the current table or a specified table is using. If you don't pass LDRIVER( ) an argument, it returns the name of the language driver of the

current table or, if no tables are open, the global language driver in use. LDRIVER( ) also returns information on Paradox and SQL databases.

The language driver associated with a table depends on the DOS code page or the BDE language driver setting that was in effect when the table was created. With *dBASE Plus*, you can choose the language driver that applies to your dBASE data in the [CommandSettings] section in the PLUS.ini file. For example, you can load a German language driver to work with a table created while that driver was active.

LINENO( )

Topic group Related topics Example

Returns the number of the current program line in the current program, procedure, or user-defined function (UDF).

Syntax

LINENO( )

Description

Use LINENO( ) to track program flow. Use it in conjunction with PROGRAM( ) to learn when a program executes a given line of code. You can also use LINENO( ) with ON ERROR to find out which line produces an error.

LINENO( ) is meaningful only when issued from within a program, procedure, or UDF. When issued in the Command window, LINENO( ) returns 0.

LINENO( ) always returns the actual program line number; the number doesn't reflect the order in which the line executes within the program.

LOGOUT( )

Topic group Related topics

LOGOUT logs out the current user and sets up a new log-in dialog.

Syntax

LOGOUT

Description

LOGOUT logs out the current user from the current session and sets up a new log-in dialog when used with PROTECT. The LOGOUT command enables you to control user sign-in and sign-out procedures. The command forces a logout and prompts for a login.

When the command is processed, a log-in dialog appears. The user can enter a group name, log-in name, and password. The PROTECT command establishes log-in verification functions and sets the user access level.

LOGOUT closes all open tables, their associated files, and program files.

If PROTECT has not been used, and no DBSYSTEM.DB file exists, the LOGOUT command is ignored.

MEMORY( )

Topic group Related topics Example

Returns the amount of currently available memory.

Syntax

MEMORY([<expN>])

<expN>

Any number, which causes MEMORY( ) to return the amount of available physical memory.

Description

Use MEMORY( ) to determine how much memory is available in the system. It returns the amount in kilobytes (1024 bytes).

In Windows, available memory is a combination of physical memory (RAM installed in the computer) and virtual memory (disk space used to simulate memory).

When called with no parameters, MEMORY( ) returns the total amount of available memory: the amount of unused physical memory plus the amount of disk space available for virtual memory. By default, Windows 95 sets no maximum for virtual memory, so MEMORY( ) will return free physical memory plus free disk space on the hard drive used for virtual memory. On Windows NT, the size of the paging file used for virtual memory is set to a reasonable size.

When called with any numeric parameter, MEMORY( ) returns the amount of free physical memory. The amount of free physical memory can vary greatly, depending on what the system is doing or has just finished doing. For example, you may have more free physical memory right after viewing and dismissing a dialog box, since the memory that was used to display the dialog box is momentarily unallocated.

dBASE Plus's About dialog box displays the amount of free memory in bytes.

MESSAGE( )

Topic group Related topics Example

Returns the error message of the most recent *dBASE Plus* error.

Syntax

MESSAGE( )

Description

Use MESSAGE( ) with other error-trapping commands and functions, such as ON ERROR, RETRY, and ERROR( ), to substitute specific responses and actions for *dBASE Plus* default responses to errors.

MESSAGE( ) is initially set to an empty string. MESSAGE( ) returns an error message when an error occurs, and remains set to that error message until one of the following happens:

- Another error occurs
- RETRY is issued
- The subroutine in which the error occurs completes execution

To learn the BDE error message of the last BDE error generated by the current table, use DBMESSAGE( ).

See the table in the description of ERROR( ) that compares CERROR( ), ERROR( ), MESSAGE( ), DBERROR( ), DBMESSAGE( ), SQLERROR( ), and SQLMESSAGE( ).

NETWORK( )

Topic group Related topics Example

Returns `true` if *dBASE Plus* is running on a system in which a local area network (LAN) card or other multiuser system card has been installed.

Syntax

NETWORK( )

Description

Use NETWORK( ) to determine if a program might be running in a network environment. For example, your program might need to do something in a network environment that it doesn't need to do in a single-user environment, such as issue USE with the EXCLUSIVE option.

NETWORK( ) returns .T. if a network card is installed; it doesn't determine whether a user is currently running *dBASE Plus* in a network environment. To determine whether a user is actually working in a network environment, use ID( ).

OEM( )

Topic group Related topics Example

Returns a character string using the current global language driver that is the equivalent of an ANSI character expression.

Syntax

OEM(<expC>)

<expC>

The ANSI character expression to convert into characters in the global language driver.

Description

OEM( ) is the inverse of ANSI( ). For more information, see ANSI( ).

ON ERROR

Topic group Related topics Example

Executes a specified statement when an error occurs.

Syntax

ON ERROR [<statement>]

<statement>

The statement to execute when an error occurs. ON ERROR without a <statement> option disables any previous ON ERROR <statement>.

Description

Use ON ERROR as a global error handler for unexpected conditions. For localized error handling—that is, for situations where you expect something might fail, like trying to open a file—use TRY...ENDTRY instead. ON ERROR also acts as a global CATCH; if there is no CATCH for a particular class of exception, an error occurs, which can be handled by ON ERROR.

When ON ERROR is active, *dBASE Plus* doesn't display its default error dialog; it executes the specified <statement>. To execute more than one statement when an error occurs, make the

<statement> DO a program file, or call a function or method. In either case, the code that is executed in response to the error is known as the ON ERROR handler.

The ON ERROR handler usually uses the ERROR( ), MESSAGE( ), PROGRAM( ), and LINENO( ) functions to determine what the error is and where it occurred. In most applications, the only safe response to an unexpected condition is to log the error and quit the application. In some cases, you may be able to fix the problem and use the RETRY command to retry the statement that caused the error; or RETURN from the ON ERROR handler, which skips the statement that caused the error and executes the next statement.

While *dBASE Plus* is executing an ON ERROR statement, that particular ON ERROR <statement> statement is disabled. Thus, if another error occurs during the execution of <statement>, *dBASE Plus* responds with its default error dialog. You can, however, set another ON ERROR handler inside a routine called with ON ERROR.

SET("ON ERROR") returns the current ON ERROR <statement>.

Avoid using a dBL command recursively with ON ERROR.

ON NETERROR

Topic group Related topics Example

Executes a specified command when a multiuser-specific error occurs.

Syntax

ON NETERROR [<command>]

<command>

The command to execute when a multiuser-specific error occurs. To execute more than one command when such an error occurs, issue ON NETERROR DO <filename>, where <filename> is a program or procedure file containing the sequence of commands to execute. ON NETERROR without a <command> option disables any previous ON NETERROR <command> statement.

Description

Use ON NETERROR to control a program's response to multiuser-specific errors. For example, in a multiuser environment on a local area network (LAN), an error can occur when two users attempt to alter the same record in a shared table at the same time, or when one user attempts to open a shared table that another user already has open for exclusive use.

ON NETERROR is similar to ON ERROR, except that ON ERROR responds to all run-time errors regardless of whether they're multiuser-specific. You can use ON ERROR to handle both single-user and multiuser errors, or you can use ON NETERROR to handle just multiuser errors. If you issue both ON ERROR and ON NETERROR, then ON ERROR responds to just single-user errors, leaving ON NETERROR to respond to multiuser errors.

While *dBASE Plus* is executing an ON NETERROR command, that particular ON NETERROR <command> statement is disabled. Thus, if another multiuser-specific error occurs during the execution of <command>, *dBASE Plus* responds with its default error messages. You can, however, set another ON NETERROR condition inside a subroutine called with ON NETERROR.

You should avoid using a dBL command recursively with ON NETERROR.

PROGRAM( )

Topic group Related topics Example

Returns the name of the currently executing program, procedure, or user-defined function (UDF).

Syntax

PROGRAM([<expN>])

<expN>

Any number.

Description

PROGRAM( ) returns the name of the lowest level executing subroutine—program, procedure, or UDF. PROGRAM( ) returns an empty string ("") when no program or subroutine is executing.

PROGRAM(expN) returns the full path name of the program that is currently running, which may be different from the name of the lowest level executing subroutine. This is shown in the following example.

```
SET PROCEDURE TO program1
** Inside PROGRAM1.PRG is PROCEDURE procedure1
** If procedure1 is running, note the following:
? PROGRAM( ) returns PROCEDURE1
? PROGRAM(expN) returns C:\Program Files\dBASE\Plus\My Programs\PROGRAM1.PRG.
```

You can issue PROGRAM( ) in the Command window if a program is suspended with SUSPEND. For example, if Program A calls Procedure B, and Procedure B is suspended, issuing PROGRAM( ) in the Command window returns the name of Procedure B; issuing PROGRAM(expN) in the Command window returns the full path name of the file containing Procedure B.

You can also use PROGRAM( ) with ON ERROR and LINENO( ) to identify the subroutine that was executing and the exact program line number at which the error occurred.

PROGRAM( ) returns the name of the subroutine in uppercase letters. PROGRAM( ) doesn't include a file-name extension even if the subroutine is a separate file, while PROGRAM(expN) always includes a file-name extension.

PROTECT

Topic group Related topics

Creates and maintains DBF table security.

Syntax

PROTECT

Description

This command is issued within *dBASE Plus* by the database administrator, who is responsible for data security. PROTECT works in a single user or multiuser environment.

PROTECT is optional. Once you create table security, you may force all users to login when *dBASE Plus* or your PLUS.exe is started, or require login only when attempting to open an encrypted table.

This command displays a multi-page dialog. This dialog is the same dialog that is displayed when choosing *dBASE Plus* table security in the File | Database Administration dialog box. The

first time you use PROTECT, the system prompts you to enter and confirm an administrator password.

Warning!

Remembering the administrator password is essential. You can access the security system only if you can supply the password. Once established, the security system can be changed only if you enter the administrator password when you call PROTECT. Keep a hard copy of the database administrator password in a secured area. There is no way to retrieve a password from the system.

Once you enter the administrator password, you may setup and modify DBF table security.

The DBSYSTEM.DB file

PROTECT builds and maintains a password system file called DBSYSTEM.DB, which contains a record for each user who accesses a PROTECTed system. Each record, called a user profile, contains the user's log-in name, account name, password, group name, and access level. When a user attempts to start *dBASE Plus* (if *dBASE Plus* is configured to require a log-in to start the program), or attempts to access an encrypted table (if *dBASE Plus* is configured to require a log-in when an encrypted table is accessed), *dBASE Plus* looks for a DBSYSTEM.DB file. You can specify a location for this file in the [CommandSettings] section of PLUS.ini:

```
DBSYSTEM=C:\Program Files\dBASE\Plus\BIN
```

If there is no DBSYSTEM entry in PLUS.ini, *dBASE Plus* looks for the file in the same directory in which PLUS.exe is located. If it finds the file, it initiates the log-in process. If it does not find the file, there is no log-in process.

DBSYSTEM.DB is maintained as an encrypted file. Keep a record of the information contained in DBSYSTEM.DB, as well as a current backup copy of the file. If the DBSYSTEM.DB file is deleted or damaged and no backup is available, the database administrator will need to reinitialize PROTECT using the same administrator password and group names as before, or the data will be unrecoverable.

RESUME

Topic group Related topics

Restarts program execution at the command line following the one at which program execution was suspended.

Syntax

RESUME

Description

RESUME causes *dBASE Plus* to resume execution of a program that is suspended. You can suspend program execution by issuing SUSPEND. If you have not assigned a value to ON ERROR, you can also choose to suspend a program when an error occurs.

To restart program execution, enter RESUME in the Command window. The program file resumes execution at the line immediately following the line that caused it to become suspended. If you want to re-execute the line that caused an error, perhaps because you fixed the condition that caused the error, retype the program line at the command line before issuing RESUME.

RETRY

Topic group Related topics Example

Returns control from a subroutine to the command line of the calling routine or Command window that called the subroutine.

Syntax

RETRY

Description

Use RETRY to re-execute a command—for example, one that resulted in an error. RETRY returns program control to the calling command. RETRY clears the memory variables created by the subroutine.

RETRY is valid only in program files.

You can use RETRY with ON ERROR to give the user more chances to resolve an error condition. Using RETRY with ON ERROR resets ERROR( ) to zero.

SET ENCRYPTION

Topic group Related topics

Establishes whether a newly created dBASE table is encrypted if PROTECT is used.

Syntax

SET ENCRYPTION ON | off

Default

The default for SET ENCRYPTION is ON.

Description

This command determines whether copied dBASE tables (that is, tables created through the COPY, JOIN, and TOTAL commands) are created as encrypted tables. An encrypted table contains data encrypted into another form to hide the contents of the original table. An encrypted table can only be read after the encryption has been deciphered or copied to another table in decrypted form.

To access an encrypted table, you must enter a valid user name, group name, and password after the login screen prompts. Your authorization and access level determine whether you can or cannot copy an encrypted table. After you access the table, SET ENCRYPTION OFF to copy the table to a decrypted form. You need to do this if you wish to use EXPORT, COPY STRUCTURE EXTENDED, MODIFY STRUCTURE, or options of the COPY TO command.

Note

Encryption works only with dBASE (.DBF) tables. Encryption works only with PROTECT. If you do not enter *dBASE Plus* or access the table through the log-in screen, you will not be able to use encrypted tables.

All encrypted tables used concurrently in an application must have the same group name.

Encrypted tables cannot be JOINed with unencrypted tables. Make both tables either encrypted or unencrypted before JOINing them.

You can encrypt any newly created table by assigning the table an access level through PROTECT.

SET ERROR

Topic group Related topics Example

Specifies one character expression to precede error messages and another one to follow them.

Syntax

SET ERROR TO
[<preceding expC> [, <following expC>]]

<preceding expC>

An expression of up to 33 characters to precede error messages. *dBASE Plus* ignores any characters after 33.

<following expC>

An expression of up to 33 characters to follow error messages. *dBASE Plus* ignores any characters after 33. If you want to specify a value for <following expC>, you must also specify a value or empty string ("") for <preceding expC>.

Default

The default for the message that precedes error messages is "Error: ". The default for the message that follows error messages is an empty string. To change the default, set the ERROR parameter in PLUS.ini, using the following format:

```
ERROR = <preceding expC> [, <following expC>]
```

Description

Use SET ERROR to customize the beginnings and endings of run-time error messages. SET ERROR TO without an argument resets the beginnings and endings to the default values.

SET ERROR is similar to ON ERROR; both can be used to customize error messages. SET ERROR, however, can only specify expressions to precede and follow a standard *dBASE Plus* error message, while ON ERROR can specify the message itself. Also unlike ON ERROR, SET ERROR can't call a procedure that carries out a series of commands.

SET LDCHECK

Topic group Related topics Example

Enables or disables language driver ID checking.

Syntax

SET LDCHECK ON | off

Default

The default for SET LDCHECK is ON. To change the default, set the LDCHECK parameter in PLUS.ini.

Description

Use SET LDCHECK to disable or enable *dBASE Plus*' capability to check for language driver compatibility. This capability is important if you work with dBASE tables created with different dBASE configurations or different international versions of dBASE because it warns you of conflicting language drivers.

Language drivers determine the character set and sorting rules that *dBASE Plus* uses, so if you create a dBASE table with one language driver and then use that file with a different language

driver, some of the characters will appear incorrectly and you may get incorrect results when querying data.

SET LDCONVERT

Topic group Related topics

Determines whether data read from and written to character and memo fields is transliterated when the table character set does not match the global language driver.

Syntax

SET LDCONVERT ON | off

Default

The default for SET LDCONVERT is ON. To change the default, set the LDCONVERT parameter in PLUS.ini.

Description

Use SET LDCONVERT to determine whether the contents of character and memo fields in tables created with a given language driver in effect, are converted to match the language driver in effect at the time the fields are read or written to.

Language drivers determine the character set and sorting rules that *dBASE Plus* uses, so if you create a dBASE table with one language driver and then use that file with a different language driver, some of the characters will appear incorrectly and you may get incorrect results when querying data.

In general, SET LDCONVERT should be ON to insure that dBASE behaves as expected when using data created under disparate language drivers.

SQLERROR( )

Topic group Related topics Example

Returns the number of the last server error.

Syntax

SQLERROR( )

Description

Use SQLERROR( ) to determine the error number of the last server error. To learn the text of the error message itself, use SQLMESSAGE( ).

See the table in the description of ERROR( ) that compares ERROR( ), MESSAGE( ), DBERROR( ), DBMESSAGE( ), SQLERROR( ), SQLMESSAGE( ), and CERROR( ).

SQLEXEC( )

Topic group Example

Executes an SQL statement in the current database or on specified dBASE or Paradox tables.

Syntax

SQLEXEC(<SQL statement expC> [,<Answer table expC>])

<SQL statement expC>

A character string that contains an SQL statement. The SQL statement must follow server-specific dialect rules for the current database and must be enclosed in quotes. For Paradox and dBASE tables, the dialect is the same as that used by the InterBase database server, which is ANSI-compliant. Character strings and SQL or BDE reserved words contained within the SQL statement must also be enclosed in either single or double quotation marks. (Single quotation marks are normally used.).

<Answer table expC>

Paradox or DBF table that stores the data returned by an SQL SELECT statement; must also be in quotes. If you specify a file without including its path, *dBASE Plus* creates the file in the current directory, then in the path you specify with SET PATH. If you specify a file without including its extension, *dBASE Plus* assumes the default table type specified with the SET DBTYPE command. If you don't specify a table name, *dBASE Plus* creates a table named Answer with the extension defined by the current DBTYPE setting.

You can also specify the name of an already open database (defined for a file directory location only) as a prefix (enclosed in colons) to the name of the answer table, that is, :database name:table name. You cannot specify the location of an answer table on a database server.

Description

SQLEXEC( ) executes an SQL statement in the current database set by SET DATABASE, or if a database is not set, on tables in the current or a specified directory. (You can preface the name of a table with its directory location or specify an already open database by enclosing the database name in colons, for example, :database name:table name. If you're using Borland SQL Link to connect to a database server, *dBASE Plus* passes the SQL statement you specify directly to the database server where the database selected by SET DATABASE resides.

When an SQL statement contains SQL or BDE reserved words and you are executing the statement on DBF or Paradox tables, you need to enclose the reserved words in single(‘) or double (") quotation marks and use SQL table aliases (different than the aliases associated with dBASE tables) to identify fields, for example:

```
SELECT * FROM company.dbf b WHERE b."CHAR" = 'element'
```

You can use table aliases to qualify fields specified in the SELECT, WHERE, GROUP BY, or ORDER BY clauses of SELECT statements. This is particularly useful when querying data from more than one table.

SQLEXEC( ) returns error codes with the same values as those returned by ERROR( ) and MESSAGE( ); a value of zero indicates that no error occurred as a result of the statement's execution. If an error occurs, you can use DBERROR( ) and DBMESSAGE( ) functions to return BDE errors or use the SQLERROR( ) and SQLMESSAGE( ) functions to obtain information directly from the database server about the cause of an error. (Also, the ERROR( ) function returns an error code of 240 if a server error occurs.)

SQLMESSAGE( )

Topic group Related topics Example

Returns the most recent server error message.

Syntax

SQLMESSAGE( )

Description

Use SQLMESSAGE( ) to determine the error message of the last server error. To learn the error code, use SQLERROR( ).

SUSPEND

Topic group Related topics Example

Suspends program execution, temporarily returning control to the Command window.

Syntax

SUSPEND

Description

SUSPEND lets you interrupt program execution at a specific point, a break point. The program remains suspended until you issue RESUME or CANCEL, or until you exit *dBASE Plus*. If you issue RESUME, the program resumes from the break point. If you issue CANCEL, *dBASE Plus* cancels program execution.

While a program is suspended, you can enter commands in the Command window. For example, you can check and change the status of files, memory variables, SET commands, and so on; however, *dBASE Plus* ignores any changes you make to the program while it is suspended. If you want to correct a suspended program, issue CANCEL, edit the program, and then run it again.

If you initialize memory variables in the Command window while a program is suspended, *dBASE Plus* makes them private at the program level that suspension occurred.

You should not return to a suspended program by issuing DO <filename> in the Command window. If you do so, you will end up with "nested" SUSPEND statements, and may not know that a program is still suspended. If you want to run a suspended program from the beginning, issue CANCEL and then DO <filename>.

The *dBASE Plus* Debugger allows for more complex break points than using SUSPEND.

USER( )

Topic group Related topics

Returns the login name of the user currently logged in to a protected system.

Syntax

USER( )

Description

The USER( ) function returns the log-in name used by an operator currently logged in to a system that uses PROTECT to encrypt files. On a system that does not use PROTECT, USER( ) returns an empty string.

VERSION( )

Topic group Related topics Example

Returns the name and version number of the currently running copy of dBASE or *dBASE Plus*.

Syntax

VERSION([<expN>])

<expN>

Any number, which causes VERSION( ) to return extended version information.

Description

Although you may be able to use VERSION( ) in programs to take advantage of version-specific features, the most common use of VERSION( ) is to get the exact build number of your copy of *dBASE Plus* to see if you have the latest build. When called with no parameters, VERSION( ) returns a string like:

```
dBASE Plus 2.0
```

with the product name and the version number. If you pass a number, for example VERSION(1), you will get extended build information, like:

```
dBASE Plus 2.0 b1672 (04/03/2002-EN020403)
```

which adds the build number after the "b" and the identifier and date of the language resource for that copy of *dBASE Plus*. If you pass the number .89, you will get the build information for the Borland Database Engine used, for example,

```
BDE version: 05/02/02
```

If you are running a *dBASE Plus* executable, the word "Runtime" appears in the string; for example:

```
dBASE Plus 2.0 Runtime
dBASE Plus 2.0 Runtime b1672 (04/03/2002-EN020403)
```

Extending dBASE Plus

Extending *dBASE Plus* with DLLs, OLE and DDE

Topic group

The classes and elements described in this section of the Help file allow you to extend *dBASE Plus* to work with Dynamic Link Libraries (DLLs) and other Windows resources, as well as communicate directly with other Windows programs through both the Object Linking and Embedding (OLE) and Dynamic Data Exchange (DDE) mechanisms.

class DDELink

Topic group Related topics Example

Initiates and controls a DDE link between *dBASE Plus* and a server application, allowing *dBASE Plus* to send instructions and data-exchange requests to the server.

Syntax

[<oRef> =] new DDELink( )

<oRef>

A variable or property in which to store a reference to the newly created DDELink object.

Properties

The following tables list the properties, events, and methods of the DDELink class.

Property	Default	Description
baseClassName	DDELINK	Identifies the object as an instance of the DDELink class

className	(DDELINK)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
server		The name of the server you specified with the *initiate*( ) method
timeout	1000	Determines the amount of time in milliseconds that *dBASE Plus* waits for a transaction before returning an error
topic		The name of the topic you specified with the *initiate*( ) method

Event	Parameters	Description
onNewValue	<item expC>, <value expC>	When an item in the server application changes

Method	Parameters	Description
advise( )	<item expC>	Requests that the server notify the client when the item in the server changes
execute( )	<cmd expC>	Sends instructions to the server in its own language
initiate( )	<server expC>, <topic expC>	Starts a conversation with a DDE server application
peek( )	<item expC>	Retrieves a data item stored by the server
poke( )	<item expC>, <value exp>	Sends a data item to the server
reconnect( )		Restores a DDE link that was terminated with *terminate*( )
release( )		Explicitly removes the DDELink object from memory
terminate( )		Terminates the link with the server application
unadvise( )	<item expC>	Asks the server to stop notifying the DDELink object when an item in the server changes

Description

Use a DDELink object to open a channel of communication (known as a DDE link) between *dBASE Plus* and an external Windows application (known as a server).

You can exchange data and instructions through this link, making the two applications work together. For example, you could use a DDELink object to open, send data to, format and print a document in your word processor, or open a spreadsheet and then exchange and update table data. You can also run separate *dBASE Plus* sessions and use one as a DDE client and the other as a server (as shown in the class DDELink and class DDETopic examples).

A DDE link is established with the *initiate*( ) method. If the server application isn't already running, *initiate*( ) attempts to start it. If the link attempt is unsuccessful, *initiate*( ) returns *false*.

class DDETopic

Topic group Related topics Example

Determines the actions taken when *dBASE Plus* receives requests from a DDE client.

Syntax

[<oRef> =] new DDETopic(<topic expC>)

<oRef>

A variable or property in which to store a reference to the newly created DDETopic object. This object reference must be returned by the *_app* object's *onInitiate* event handler.

<topic expC>

The name of the topic to which the DDETopic object responds.

Properties

The following tables list the properties, events, and methods of the DDETopic class.

Property	Default	Description
baseClassName	DDETOPIC	Identifies the object as an instance of the DDETopic class
className	(DDETOPIC)	Identifies the object as an instance of a custom class. When no custom class exists, defaults to baseClassName
topic		The DDETopic object's topic

Event	Parameters	Description
onAdvise	<item expC>	After an external application creates a hot link
onExecute	<cmd expC>	When a client application sends a command to *dBASE Plus*
onPeek	<item expC>	When the client requests a value from *dBASE Plus*
onPoke	<item expC>, <value expC>	When the client sends a new value for a *dBASE Plus* item
onUnadvise	<item expC>	After a client removes a hot link from a particular item

Method	Parameters	Description
notify( )	<item expC>	Notifies all interested client applications that a *dBASE Plus* item was changed
release( )		Explicitly removes the DDETopic object from memory

Description

Use a DDETopic object to determine what *dBASE Plus* does for a client application when *dBASE Plus* is the server in a DDE link. *dBASE Plus* may act as a DDE server for one or more topics; client applications must specify the topic they are interested in when they create the DDE link. (A client application may link to more than one topic at a time.)

As a server application, *dBASE Plus* accepts either a generic command string, or a named item-value pair from a client application. It must respond to requests for a named data item, and notify interested client applications when a item value changes.

You usually create a DDETopic object in an initiation-handler routine, which you assign to the *onInitiate* event of the *_app* object. The initiation-handler executes when a client application requests a DDE link with *dBASE Plus* and no DDETopic object exists in memory for the desired topic. Each newly-created DDETopic object must be returned by the *onInitiate* event handler; that object is automatically stored internally to respond to client requests on that topic.

When *dBASE Plus* acts as a DDE server, the topics and item names it maintains internally are not case-sensitive; topic and item names that match existing names (regardless of case) will be changed to those names before being passed to events. When writing event handlers, be aware that the names may vary in case—use UPPER( ) or LOWER( ) to make the names consistent in the program logic.

For information on using *dBASE Plus* as a client application, see class DDELink.

class OleAutoClient

Topic group Example

Creates an OLE2 controller that attaches to an OLE2 server.

Syntax

[<oRef> =] new OleAutoClient(<server expC>)

<oRef>

A variable or property in which you want to store a reference to the newly created OleAutoClient object.

<server expC>

The name of the OLE Automation server. The name is of the form, "app.object"; for example, "word.application"

Properties

The properties, events, and methods of each instance of the OleAutoClient class depend on the attached OLE automation server.

Description

OLE automation allows you to control another application, an OLE automation server, through an OLE automation client. For example, with a full-featured word processor as an OLE automation server, you could do the following all on the server machine:

- Start the word processor
- Open an order form
- Fill in data that was entered in a client browser
- Fax that order form to a customer
- Close the word processor

With *dBASE Plus* as the host for the OLE automation client, you could control the entire process from a browser. You don't even need the word processor to be installed on the *dBASE Plus* client, just on the *dBASE Plus* server machine.

dBASE Plus's dynamic object model is a natural host for OLE automation clients. Because there is no need to declare the capabilities of the OLE automation server as you would with a statically linked language, you can specify any OLE Automation server at run time, and use whatever capabilities it has.

Once you create the OleAutoClient object, the properties, events, and methods the OLE automation server provides are accessed through the OleAutoClient object, just as with stock *dBASE Plus* objects. You can also *inspect*( ) the OleAutoClient object's properties.

advise( )

Topic group Related topics Example

Creates a DDE hot link to enable the passing of notification messages whenever a specified topic item changes in the server application.

Syntax

<oRef>.*advise*(<item expC>)

<oRef>

A reference to the DDELink object that wants to get notified.

<item expC>

The name of the topic item you want to monitor.

Property of

DDELink

Description

Use the *advise*( ) method to create a hot link to an item in a server topic. A hot link requires the server application to notify *dBASE Plus* when the value of a specified item changes.

A server topic can be anything that the server application understands, but it is most often a document, spreadsheet, or database that you specified when you called the *initiate*( ) method to create the DDE link with the external application. If, for example, you use *initiate*( ) to open a spreadsheet application as the server application and a worksheet as a topic, you could specify any item or range in the worksheet as the topic item you want monitored for changes.

The DDELink object's *onNewValue* event will fire with the item name and new value as parameters whenever the named item changes.

CALLBACK

Topic group

CALLBACK allows you to write a function or method in dBL code and setup a pointer to it that can be passed to an external program - such as Windows or the BDE or some other 3rd party software that supports callbacks.

Callbacks can be used to notify your program of specific events occurring with an external program OR to allow your program to modify what an external program is doing.

Syntax

CALLBACK <return type> <function name>([<param type>,...])

[FROM <filename>]

[OBJECT <oRef>]

<return type>

EXTERN type returned by function

<function name>

Name of dBL function or method to be made callable From external program

<param type>,...

Zero or more EXTERN types specifying the parameter types expected by the dBL function or method

<filename>

dBL source file containing function or method

<oRef>

Object reference of object containing dBL method

See Also

GETCALLADDRESS

RELEASE CALLBACK

execute( )

Topic group Related topics Example

Sends a command string to a DDE server application in its own language.

Syntax

<oRef>.*execute*(<cmd expC>)

<oRef>

A reference to the DDELink object that has the link.

<cmd expC>

Command or macro to send to the DDE server application.

Property of

DDELink

Description

Use the *execute*( ) method to send commands to DDE server applications.

The command string must be in the language of the server application, or any other string that the server expects.

Be sure to enclose commands in the delimiters required by the server application. For example, Quattro Pro commands must be enclosed in braces ({ }), while Word for Windows 95 commands are enclosed in brackets ([ ]). Some applications accept multiple commands separated by brackets. For information, consult your DDE server documentation.

Before you can send a command string to a server, you must open the server application, open the document, and establish a DDE link. For information on establishing DDE links, see *initiate*( ).

EXTERN

Topic group Related topics Example

Declares a prototype for a non-dBL function contained in a DLL file.

Syntax

EXTERN [CDECL | PASCAL | STDCALL] <return type> <function name>
([<parameter type> [, <parameter type> ...]])
<filename>

or

EXTERN [CDECL | PASCAL | STDCALL] <return type> <user-defined function name>
([<parameter type> [, <parameter type> ...]])

<filename>
FROM <export function name expC> | <ordinal number expN>

Because you create a function prototype with EXTERN, parentheses are required as with other functions.

CDECL | PASCAL | STDCALL

Sets the function calling convention. The default is STDCALL.

<function name>

The export name of the function. The export name of an external function is contained in the DEF file associated with the DLL file that holds the function, or explicitly exported in the source code.

Note

With STDCALL and CDECL, function names are case-sensitive.

<return type> and <parameter type>

A keyword representing the data type of the value returned by the function, and the data type of each argument you send to the function, respectively. The following tables list the keywords you can use.

Parameters or return values

Keyword	as pointer	dBASE Plus data type	Data Type size
CINT	CPTR CINT	Numeric	4 bytes (32 bits)
CLONG	CPTR CLONG	Numeric	4 bytes (32 bits)
CSHORT	CPTR CSHORT	Numeric	2 bytes (16 bits)
CCHAR		Numeric	1 byte (8 bits)
	CSTRING	String	Null-terminated
CHANDLE		Numeric	4 bytes (32 bits)
CUINT	CPTR CUINT	Numeric	4 bytes (32 bits)
CULONG	CPTR CULONG	Numeric	4 bytes (32 bits)
CUSHORT	CPTR CUSHORT	Numeric	2 bytes (16 bits)
CUCHAR		Numeric	1 byte (8 bits)
CFLOAT	CPTR CFLOAT	Numeric	4 bytes (32 bits)
CDOUBLE	CPTR CDOUBLE	Numeric	8 bytes (64 bits)
CLDOUBLE	CPTR CLDOUBLE	Numeric	10 bytes (80 bits)
CLOGICAL	CPTR CLOGICAL	Logical	4 bytes (32 bits)
CVOID		none	N/A

Parameters only

Keyword	as pointer	dBASE Plus data type	Data Type size
	CPTR	String	
...		N/A	

In most cases, if the function expects a pointer as a parameter, *dBASE Plus* will pass a pointer to the value. If a function returns a pointer, *dBASE Plus* will get the value at the pointer and convert it into the appropriate *dBASE Plus* data type.

CCHAR is actually a numeric data type, represeting a single-byte value. When passing a CCHAR parameter, you may pass a string; *dBASE Plus* sends the ASCII value of the first character in the string. The return value is always a number.

If the function has no parameters or returns no value, declare the data type as CVOID.

You may use the ... parameter declaration if the calling convention is STDCALL to designate a variable number of parameters.

Using strings

dBASE Plus is a Unicode application; using strings is more complicated in 32-bit programming than in 16-bit programming. Many Windows API functions have both an A (ANSI) and W (wide-character) version. The A functions use single-byte characters, and the W versions use double-byte characters. When calling the A version of a function, always use CSTRING. When calling the W version of a function, always use CPTR. When you use CSTRING, *dBASE Plus* will convert the Unicode string it uses into an ANSI string when passing it to the function. After the function call, it converts the ANSI string back into Unicode. With CPTR, no conversion takes place.

When strings (CSTRING or CPTR) are used as parameters, they are always passed and read back by length. You may include null characters in the string. When a return value is declared as CSTRING, it is read as null-terminated. You cannot use CPTR as a return value.

Using structures

If the function expects a pointer to a structure, declare the parameter as CPTR. Use a string to represent the structure contents. Because *dBASE Plus* strings are Unicode, you must use the String object's *getByte( )* and *setByte( )* methods to read and write the structure, byte-by-byte.

<filename>

The name of the DLL file in which the external function is stored. If the extension is omitted, the default is DLL. The file name of any DLL that you load in memory must be unique; for example, you can't load SCRIPT.DLL and SCRIPT.FON into memory concurrently, even though they have different file-name extensions.

If the DLL file is not already loaded into memory, EXTERN loads it automatically. If the DLL file is already in memory, EXTERN increments the reference counter. Therefore, it isn't necessary to execute LOAD DLL before using EXTERN.

The reference counter is incremented only the first time, regardless of how many times you execute the LOAD DLL and EXTERN statements.

You may include a path in <filename>. If you omit the path, *dBASE Plus* looks in the following directories for the DLL by default:

1. The directory containing PLUS.exe, or the directory in which the .EXE file of your compiled application is located.
2. The current directory.
3. The 32-bit Windows system directory (for example, C:\WIN95\SYSTEM).
4. The 16-bit Windows system directory, if present (for example, C:\WINDOWS\SYSTEM).
5. The Windows directory (for example, C:\WINNT)
6. The directories in the PATH environment variable

The path specification is necessary only when the DLL file is not in one of these directories.

<user-defined function name>

The name you give to the external function instead of the export name. This is usually used to rename the A or W version of a function to the generic name. When you specify <user-defined function name> (instead of <function name>), you must use the FROM clause to identify the function in the DLL file.

FROM <export function name expC> | <ordinal number expN>

Identifies the function in the DLL file specified by <filename>. <export function name expC> identifies the function by its name. <ordinal number expN> identifies the function with a number.

Description

Use EXTERN to declare a prototype for an external function written in a language other than dBL. A prototype tells *dBASE Plus* to convert its arguments to data types the external function can use, and to convert the value returned by the external function into a data type *dBASE Plus* can use.

To call an external DLL function, first prototype it with EXTERN. Then, using the name of the function you specified with EXTERN, call the function as you would any dBL function. You must prototype an external function before you can call that function in *dBASE Plus*.

The external function may be in any 32-bit DLL, such as the Windows API or a third-party DLL file. Although most library code is contained in files with extensions of DLL, such code can be held in EXE files, or even in DRV or FON files.

GETCALLADDRESS

Topic group

returns the address to pass to an external program for a dBL Callback function or method

Syntax

GETCALLADDRESS([<expC>][<expF>])

<expC>

name of dBL function or method

<expF>

function pointer for a dBL function or method.

See Also

CALLBACK

RELEASE CALLBACK

initiate( )

Topic group Related topics Example

Starts a conversation with an external application or aliased DDE server.

Syntax

<oRef>.*initiate*(<server expC>, <topic expC>)

<oRef>

A reference to the DDELink object through which you want to initiate the DDE link.

<server expC>

The executable filename of the server application (normally the .EXE extension isn't necessary) or the alias name of a running DDE server.

<topic expC>

Name of a built-in DDE topic, document, or other topic.

Property of

DDELink

Description

Use *initiate*( ) to open a channel of communication (known as a DDE link) between *dBASE Plus* and a running external Windows application or aliased DDE server.

If you call an external application with *initiate*( ), *dBASE Plus* tries to open the application if it is not already running. *initiate*( ) returns *true* if the connection is successful, and *false* if the connection attempt fails.

To close the DDE link, use *terminate*( ).

LOAD DLL

Topic group Related topics Example

Initiates a DLL file.

Syntax

LOAD DLL [<path>] <DLL filename>

[<path>] <DLL name>

The name of the DLL file. <path> is the directory path to the DLL file in which the external function is stored.

Description

Use LOAD DLL to make the resources of a DLL file available to your application.

Note

dBASE Plus uses 32-bit DLLs only; it cannot use 16-bit DLLs.

You can also use LOAD DLL to check for the existence of a DLL file. For example, you can use the ON ERROR command to execute an error trapping routine each time the LOAD DLL command can't find a specified DLL file.

LOAD DLL does not use the *dBASE Plus* path to find DLL files. Instead, it searches the following directories:

1. The directory containing PLUS.exe, or the directory in which the .EXE file of your compiled application is located.
2. The current directory.
3. The 32-bit Windows system directory (for example, C:\WIN98\SYSTEM).

4. The 16-bit Windows system directory, if present (for example, C:\WINDOWS\SYSTEM).
5. The Windows directory (for example, C:\WINNT)
6. The directories in the PATH environment variable

A DLL file is a precompiled library of external routines written in non-dBL languages such as C and Pascal. A DLL file can have any extension, although most have extensions of .DLL.

When you initialize a DLL file with LOAD DLL, *dBASE Plus* can access its resources; however, it doesn't become resident in memory until your program or another Windows program uses its resources.

To access a DLL function, create a dBL function prototype with EXTERN. Then, using the name you specified with EXTERN, call the routine as you would any dBL function.

notify( )

Topic group Related topics Example

Notifies all interested client applications that a *dBASE Plus* item was changed.

Syntax

<oRef>.*notify*(<item expC>)

<oRef>

A reference to the DDETopic object in which the item changed.

<item expC>

Name of the item that has changed.

Property of

DDETopic

Description

Use *notify*( ) in a DDE server program to tell all interested client applications that an item in the *dBASE Plus* server was changed.

Client applications ask to be notified of changes by calling their equivalent of the DDELink object's *advise*( ) method. *dBASE Plus* automatically maintains an internal list of these clients so that when the *notify*( ) method is called, the appropriate DDE message is sent to each client, if any. For a *dBASE Plus* client, that message fires the *onNewValue* event.

onPoke event handlers often call *notify*( ) when an external application sends *dBASE Plus* a *poke* request. For example, a Quattro Pro data-exchange application might use its {POKE} command to send *dBASE Plus* a value, causing the *onPoke* event handler to execute. The *onPoke* routine could insert the value into a field, then execute *notify*( ) to inform Quattro Pro that the field changed.

When *notify*( ) is called, the *onPeek* method is called implicitly to get the value of the item to pass to the client applications.

onAdvise

Topic group Related topics Example

Event fired after an external application requests a DDE hot link to an item in a *dBASE Plus* server topic.

Parameters

<item expC>

The data item for which the external application wants to be advised when changes are made.

Property of

DDETopic

Description

Use *onAdvise* in a DDE server program to respond to a request for a hot link and to determine which *dBASE Plus* data item the link applies to. To implement a hot link in a DDETopic object, use the *notify*( ) method to advise all interested clients whenever the item changes.

dBASE Plus automatically maintains an internal list of all clients that asked to be notified on changes in a particular item (when the client calls their equivalent of the DDELink object's *advise*( ) method). This makes *onAdvise* supplemental. For example, you can track those items for which there has been a notification request. Whenever an item changes, you can call *notify*( ) only if someone has requested notification.

Item names are often held in tables or arrays to handle multiple hot links. For example, a client application might request a hot link to a field, passing the field name through the <item> parameter. Each time, the *onAdvise* event handler places the field name in an array.

The *onPoke* event handler would search this array each time a field is changed; if the name of the changed field is found in the array, the routine could call the *notify*( ) method.

onExecute

Topic group Related topics Example

Event fired when a client application sends a command string to a DDE server program.

Parameters

<cmd expC>

The command string sent by the client application.

Property of

DDETopic

Description

Use the *onExecute* property to perform an action when the client application sends a directive to *dBASE Plus*. This directive can be any string of characters.

For example, a stock trading routine might receive either of two character strings, "BUY" or "SELL". The routine could use an IF statement to perform one action or another accordingly. A web browser could take a URL as a command, and display that URL.

onNewValue

Topic group Related topics Example

Event fired when a hot-linked item in a DDE server changes.

Parameters

<item expC>

Identifies the server item that was changed.

<value expC>

The new value of the hot-linked server item.

Property of

DDELink

Description

Use *onNewValue* to perform an action when a hot-linked server item is changed. A hot link, which you create with the *advise*( ) method, tells the server to notify *dBASE Plus* when the item changes.

Note that the value of the item is always converted to a string.

onPeek

Topic group Related topics Example

Event fired when a client application tries to read an item from a DDE server application.

Parameters

<item expC>

The data item the client application wants to read.

Property of

DDETopic

Description

Use the *onPeek* property to send a value to a client application when the client application makes a *peek* request.

The *onPeek* event handler must RETURN the value of the requested item. *dBASE Plus* automatically handles the internal DDE mechanism to pass the value back to the client.

onPeek is also called implicitly by the DDETopic object's *notify*( ) method so that the DDE server can send an item's name and value when it is changed.

onPoke

Topic group Related topics Example

Event fired when a client application attempts to send a value for a DDE server item.

Parameters

<item expC>

The name of the item that identifies the value.

<value expC>

The desired value

Property of

DDETopic

Description

Use the *onPoke* event to receive a named value from a client application. For example, the <item expC> could identify a cell in a spreadsheet, and the <value exp> is the value to store in that cell. Another example would be for the <item expC> to contain a command, and the <value exp> to act as a parameter to that command.

Note

If a client established a hot link before sending the data, you should execute the *notify*( ) method in the *onPoke* event handler, thus informing the client application(s) that a change occurred.

onUnadvise

Topic group Related topics Example

Event fired after *dBASE Plus* is requested to stop notifying the client application when a *dBASE Plus* data item changes.

Parameters

<item expC>

The data item for which the external application no longer wants to be advised when changes are made.

Property of

DDETopic

Description

Use *onUnadvise* in a DDE server program to respond to a request to cancel a hot link.

dBASE Plus automatically maintains an internal list of all clients that asked to be notified on changes in a particular item, and will automatically remove a client when the client calls their equivalent of the DDELink object's *unadvise*( ) method. This makes *onUnadvise* supplemental. If you have been doing your own tracking in an *onAdvise* event, you will want to undo that in the *onUnadvise* event handler.

peek( )

Topic group Related topics Example

Retrieves a data item from a DDE server.

Syntax

<oRef>.*peek*(<item expC>)

<oRef>

A reference to the DDELink object that has the link.

<item expC>

The name of the desired item.

Property of

DDELink

Description

Use the *peek*( ) method to read data from a DDE server topic.

peek( ) takes a data item in the server topic. This item can be any single element, such as a field in a table or a cell in a spreadsheet. For example, you can read cell C2 of Page A in a Quattro Pro spreadsheet file by passing the <item> parameter "A:C2".

PLAY SOUND

Topic group Example

Plays a sound stored in a .WAV file or a binary field.

Syntax

PLAY SOUND FILENAME <filename> | ? | <filename skeleton> |
or
PLAY SOUND BINARY <binary field>

**FILENAME <filename> | ? | <filename skeleton> |
or BINARY <binary field>**

Specifies the sound file or binary field. PLAY SOUND FILENAME ? and PLAY SOUND FILENAME <filename skeleton> display a search dialog to let you select a sound file (the .WAV extension is assumed, but you can specify otherwise). PLAY SOUND BINARY <binary field> plays the sound stored in a binary field.

Description

Use PLAY SOUND in your programs to run .WAV files or audio data stored in binary fields.

poke( )

Topic group Related topics Example

Sends data to a DDE server.

Syntax

<oRef>.*poke*(<item expC>, <value expC>)

<oRef>

A reference to the DDELink object that has the link.

<item expC>

The name of the desired item.

<value expC>

The value you want to send (as a string).

Property of

DDELink

Description

Use the *poke*( ) method to write data to a DDE server application.

For example, a data-exchange program could start a session in Quattro Pro, open one of its spreadsheet files, and use *poke*( ) to write a value into one of its cells.

reconnect( )

Topic group Related topics Example

Attempts to restart a terminated conversation with a DDE server application.

Syntax

<oRef>.*reconnect*( )

<oRef>

A reference to the DDELink object that had the link.

Property of

DDELink

Description

Use *reconnect*( ) to restore a DDE link that was terminated with the *terminate*( ) method. It returns *true* if successful; *false* if not.

When you terminate a DDE link with *terminate*( ), you can restore it with *reconnect*( ). When you terminate a link with the *release*( ) method, however, the link can't be restored and must be recreated with another DDELink object.

RESOURCE( )

Topic group Related topics Example

Returns a character string from a DLL file.

Syntax

RESOURCE(<resource id>, <DLL filename expC>)

<resource id>

A numeric value that identifies the character string resource.

<DLL filename expC>

The name of the DLL file.

Description

Use RESOURCE( ) to generate a character string from a resource in a DLL file. The character string must be less than 32K; a character string longer than this is truncated.

RESOURCE( ) is often useful for internationalizing applications without changing program code. For example, you can store in a DLL file all character strings that might need translation from one language to another, and your application can retrieve them at run time with the RESOURCE( ) function. To modify the application for another language, translate the strings and store them in a new DLL file, in the same order and with the same resource IDs as their counterparts in the original DLL file. Then substitute the new DLL for the original one.

RELEASE DLL

Topic group Related topics Example

Deactivates DLL files.

Syntax

RELEASE DLL <DLL filename list>

Description

Use RELEASE DLL when you debug a DLL file or a *dBASE Plus* application. For example, you must deactivate a DLL file and activate it again each time you change one of its routines.

A DLL file is a precompiled library of external routines written in non-dBL languages such as C and Pascal. A DLL file can have any extension, although most have extensions of .DLL. You activate a DLL file with the EXTERN or LOAD DLL commands.

RELEASE CALLBACK

Topic group

RELEASE CALLBACK allows you release an object instantiated by the CALLBACK function.

Syntax

RELEASE CALLBACK <function name>
    [OBJECT <oRef>]

RELEASE CALLBACK [ALL]

<function name>

Name of dBL function or method to release the callback

<oRef>

Object reference of object containing dBL method/function.

See Also

GETCALLADDRESS

CALLBACK

RESTORE IMAGE

Topic group Related topics Example

Displays an image stored in a file or a binary field.

Syntax

RESTORE IMAGE FROM
 <filename> | ? | <filename skeleton> | BINARY <binary field>
[TIMEOUT <expN>]
[TO PRINTER]
[[TYPE] <*file type*>]

FROM <filename> | ? | <filename skeleton> | BINARY <binary field>

Identifies the file or binary field to restore the image from. RESTORE IMAGE FROM ? and RESTORE IMAGE FROM <filename skeleton> display the Open Source File dialog box, which lets the user select a file. <filename> is the name of an image file; RESTORE IMAGE assumes

a .BMP extension and file type unless you specify otherwise. If you specify a file without including its path, *dBASE Plus* looks for the file in the current directory, then in the path you specify with SET PATH. RESTORE IMAGE FROM BINARY <binary field> displays the image stored in a binary field. You store an image in a binary field with the REPLACE BINARY command.

TIMEOUT <expN>

Specifies the number of seconds the image is displayed onscreen.

TO PRINTER

Sends the image to the printer as well as to the screen.

[TYPE] <file type>

Specifies a bitmap image format, and assumes a .BMP format and filename extension if none is given. The word TYPE is optional. See class Image for a list of image formats supported by *dBASE Plus*.

Description

Use RESTORE IMAGE to display a bitmap image that was generated and saved in any of the supported image file formats. The image is displayed in a window.

server

Topic group Related topics

Holds the name of a DDE server application.

Property of

DDELink

Description

The read-only *server* property contains the DDE service name of server application that you established a DDE link to.

SET HELP

Topic group

Determines which Help file (.HLP) or (.CHM) the dBASE Plus Help system uses.

Syntax

SET HELP TO [<help filename> | ? | <help filename skeleton>]

<help filename> | ? | <help filename skeleton>

Identifies the Help file to activate. ? and <filename skeleton> display a dialog box, from which you can select a file. If you specify a file without including its extension, dBASE Plus assumes .HLP.

Description

Use SET HELP TO to specify which Help file to use when the dBASE Plus Help system is activated.

The Help file is opened automatically when you start dBASE Plus if you place the file in the dBASE Plus home directory. SET HELP TO closes any open Help file before it opens a new file.

Examples

To display a dialogue box from which to select an available Help file, issue the following command:

```
SET HELP TO ?
```

or

```
SET HELP TO *.HLP
```

To set Help to your own tailored help file:

```
SET HELP TO "C:\PROGRAM FILES\MyApplication\HELP\MyHlp.HLP"
```

Note: It's advisable to store custom help files in your application's directory, or a "HELP" sub-directory, as illustrated in the preceding command. We used the full path in this example for the sake of simplicity, since each users path will vary according to their current location. When in doubt, the full path always works. As in the above example, quotes are necessary when the path includes spaces.

To set the Help file back to the dBASE Plus default:

```
SET HELP TO
```

terminate( )

Topic group Related topics Example

Terminates a conversation with a DDE server application.

Syntax

<oRef>.*terminate*( )

<oRef>

A reference to the DDELink object whose connection to terminate.

Property of

DDELink

Description

Use *terminate*( ) to close a DDE link between *dBASE Plus* and a server application.

terminate( ) stops communication between *dBASE Plus* and the server application, but doesn't close the server application itself.

When you terminate a DDE link with *terminate*( ), you can restore it with *reconnect*. When you terminate the link with the *release*( ) method, the link can't be restored and you need to create another DDELink object again.

timeout

Topic group Related topics Example

Specifies the amount of time in milliseconds that *dBASE Plus* waits on a transaction before returning an error.

Property of

DDELink

Description

Use *timeout* to set a limit on the length of time *dBASE Plus* waits for a DDE transaction to complete successfully.

Errors sometimes occur when *dBASE Plus* tries to exchange data with a server application or send instructions to it. Each time an attempt is made, *dBASE Plus* waits for the amount of time you specify (in milliseconds) with *timeout*. When the transaction fails to complete in the allotted time, *dBASE Plus* generates an error.

When using DDE over a network, you may need to set *timeout* to a larger value.

topic

Topic group Related topics

Holds the name of the topic of a DDELink or DDETopic object.

Property of

DDELink, DDETopic

Description

The *topic* property of a DDELink object contains the name of the topic for which you established a DDE link.

The *topic* property of a DDETopic object distinguishes the object from other DDETopic objects. For example, a *dBASE Plus* server application might create two DDETopic objects, one with a topic of NASDAQ and the other with a topic of AMEX:

```
xServer1 = new DdeTopic("NASDAQ")
xServer2 = new DdeTopic("AMEX")
```

The *topic* property is read-only.

unadvise( )

Topic group Related topics Example

Asks the server to stop notifying the client when an item in the server document changes.

Syntax

<oRef>.*unadvise*(<item expC>)

<oRef>

A reference to the DDELink object that no longer wants notification.

<item expC>

The name of a topic item previously hot-linked using *advise*( ).

Property of

DDELink

Description

Use the *unadvise*( ) method to disconnect a hot link to a DDE server item. A hot link, which you create with the *advise*( ) method, provides a way for the server to inform the client when a specified item, such as a field or a spreadsheet cell, has changed.

Preprocessor

Preprocessor

Topic group

When *dBASE Plus* compiles a program file, that file is run through the preprocessor before it is actually compiled. The preprocessor is a separate built-in utility that processes the text of the program file to prepare it for compilation. Its duties include

- Stripping out comments from the program file
- Joins lines separated by the line continuation character
- Substituting macro-identifiers and macro-functions with the corresponding replacement text
- Including the text of other files in the program file
- Conditionally excluding parts of the program file so they are not compiled

The preprocessor generates an intermediate file; this is the file that is compiled by *dBASE Plus*' compiler.

While those are the mechanics of the preprocessor, the usage of the preprocessor allows you to

- Replace constants and "magic numbers" in your code with easy-to-read and easy-to-change identifiers
- Create macro-functions to replace complex expressions with parameters
- Use collections of constant identifiers and macro-functions in multiple program files
- Maintain separate versions of your programs, for example debug and production versions, in the same program files through conditional compilation

dBASE Plus' preprocessor is similar to the preprocessor used in the C language. It uses a handful of preprocessor directives to control its activities. All preprocessor directives start with the # character and each one must be on its own, single line in the script file.

#define

Topic group Related topics Example

Defines an identifier (name) for use in controlling program compilation, defining constants, or creating macro-functions.

Syntax

#define <identifier> [<replacement text>]

#define <identifier>(<parameter list>) <replacement text with parameters>

<identifier>

A name. It identifies the text to replace if <replacement text> is supplied. The name must start with an alphabetic character and can contain any combination of alphabetic or numeric characters, uppercase or lowercase letters. The identifier is not case-sensitive.

(<parameter list>)

Parameter names that correspond to arguments passed to a macro-function that you create with #define <identifier>(<parameter list>) <replacement text>. If you specify multiple parameters, separate each with a comma. There cannot be any spaces between the <identifier> and the opening parenthesis of (<parameter list>), or after any of the parameter names in the <parameter list>.

<replacement text>

The text you want to use to replace all occurrences of <identifier>. If you specify <replacement text>, the preprocessor scans each source code line for identifiers and replaces each one it

encounters with the specified replacement text. <replacement text> can be any text that is part of a dBL program, such as a string, numeric expression, or series of commands.

Description

The #define directive defines an identifier and optionally lets you replace text in a program before compilation. Each #define definition must begin on a new line and is limited to 4096 characters.

Identifiers are available only to the program in which they are defined. To define identifiers for use in multiple programs, place them in a separate file and use #include to include that file as needed.

You must define an identifier in a file with the #define directive before you can use it. Once it has been defined, you cannot #define it again; you must undefine it first with the #undef preprocessor directive.

Use the #define directive for the following purposes:

- To declare an identifier and assign replacement text to represent a constant value or a complex expression.
- To create a macro-function.
- To declare an identifier with no replacement text, so you can use it with the #ifdef or #ifndef directive.
- To declare an identifier with replacement text, so you can use it with the #if directive.

The effect of #define is similar to a word processor's search-and-replace feature. When the preprocessor encounters a #define identifier in the text of a script, it replaces that identifier with the <replacement text>. If there is no <replacement text> for that identifier, the identifier is simply removed. For example:

```
#define test 4 // Create identifier with value
? test - 3 // ( 4 - 3 ) = 1
#undef test // #undef to change definition
#define test // Create identifier with no value
? test - 3 // ( - 3 ) = -3
```

Because the preprocessor runs before a program is compiled and performs simple text substitution, the use of #define statements can in effect override memory variables, built-in commands and functions, and any other element having the same name as <identifier>. This is shown in the following examples.

```
// Overiding a variable
somevar = 25; // Creates variable
#define somevar 10; // Until further notice, "somevar" will be replaced
? somevar // Compiles argument as "10". Displays 10
#undef somevar // "somevar" no longer replaced
? somevar // Displays 25

// Overriding a function
#define cos(x) (42 + x) // Function adds 42
? cos(3) // Compiles argument as "(42 + 3)". Displays 45
```

To use #define directives in WFM and REP files generated by the Form and Report designers, place the directives in the Header section of the file so that the definitions will not be erased by the designer.

Declaring identifiers to represent constants

Assign an identifier to represent a constant value or expression when you want the preprocessor to search for and replace all instances of the identifier with the specified value or expression before compilation. When used in this manner, the identifier is known as a manifest constant. It's common practice to make the name of the manifest constant all uppercase, with underscores between words so that it stands out in code. For example:

```
#define SECS_PER_HOUR 3600 // Number of seconds per hour
```

```
#define MSECS_PER_DAY (1000*24*SECS_PER_HOUR) // Number of milliseconds per day
```

Note that when using a manifest constant to represent a numeric expression, you should place the entire expression inside parentheses. This prevents possible errors due to the precedence of operators used to evaluate expressions. For example, consider the following calculation:

```
nDays = nTimeElapsed / MSECS_PER_DAY
```

Without parentheses, this statement would compile as:

```
nDays = nTimeElapsed / 1000*24*3600
```

That's incorrect—it divides by 1000 then multiplies by 24 and 3600. (The multiplication and division operators are at the same level of precedence, so the expression is evaluated from left to right.). By placing parentheses around the manifest constant definition as shown, the statement would compile as:

```
nDays = nTimeElapsed / (1000*24*3600)
```

Because of the parentheses, the expression is evaluated correctly: the value of the constant is evaluated first, then used as the divisor.

Manifest constants streamline your code and improve its readability because you can use a single identifier to represent a frequently used constant or a complex expression. In addition, if you need to change the value of a constant in your program, you need to change only the constant definition and not every occurrence of the constant.

To replace an identifier only in parts of a program, insert #undef <identifier> into your program where you want the search-and-replace process to stop.

Creating macro-functions

When the preprocessor encounters a function call that matches a defined macro-function, it replaces the function call with the replacement text, inserting the arguments of the function call into the replacement text. This is shown in the following example.

```
#define avg(num1,num2) (((num1)+(num2))/2) // Average two numbers
...
n1=20
n2=40
? avg( n1, n2 ) // Displays 30
```

The arguments in the macro-function call are substituted exactly as they are shown in the macro-function definition. In this example, the last statement compiles as:

```
? (((n1)+(n2))/2)
```

When using a macro-function to perform calculations, always use parentheses to enclose each parameter and the entire expression in the macro-function definition as shown. If you leave them out, errors may result due to the precedence of operators, as shown in these (somewhat contrived) examples:

```
#define avg(num1,num2) (((num1)+(num2))/2)
#define badAvg(num1,num2) (num1+num2)/2
? 12 / avg( 2, 4 ) // 12/(6/2) --> displays 4
? 12 / badavg( 2, 4 ) // 12/6/2 --> displays 1
```

Unlike functions in dBL, the number of arguments passed from a macro-function call must match the number of parameters defined in your #define statement.

Declaring identifiers for conditional compilation

In addition to using identifiers for constants and macro-functions in dBL code, they are used for conditional compilation with the #if, #ifdef, and #ifndef directives.

Defining an identifier without replacement text lets you use it with the #ifdef or #ifndef directive to test if the identifier exists. When used in this manner, the existence of the identifier acts as a logical flag to either include or exclude code for compilation.

When an identifier is defined with replacement text, you can use comparison operators to check the value of the identifier in an #if directive, and conditionally compile code based on the result. You can also use #ifdef and #ifndef to test for the existence of the identifier.

Nesting preprocessor identifiers

You can nest preprocessor identifiers; that is, the replacement text for one identifier may contain other identifiers, as long as those identifiers are already defined, as shown in the following example:

```
#define SECS_PER_HOUR 3600 // Number of seconds per hour
#define MSECS_PER_DAY (1000*24*SECS_PER_HOUR) // Number of milliseconds per day
```

You cannot use the identifier being defined in its replacement text, however.

#else

Topic group Related topics

Designates an alternate block of code to conditionally compile if the condition specified by an #if, #ifdef, or #ifndef directive is *false*.

#endif

Topic group Related topics

Designates the end of an #if, #ifdef, or #ifndef directive.

#if

Topic group Related topics Example

Controls conditional compilation of code based on the value of an identifier assigned with #define.

Syntax

#if <condition>
<statements 1>

[#else
<statements 2>]

#endif

<condition>

A logical expression, using an identifier you've defined, that evaluates to *true* or *false*.

<statements 1>

Any number of statements and preprocessor directives. These lines are compiled if <condition> evaluates to *true*.

#else <statements 2>

Specifies the lines you want to compile if <condition> evaluates to *false*.

Description

Use the #if directive to conditionally compile sections of source code based on the value of an identifier. Two other directives, #ifdef and #ifndef, are also used to conditionally include or exclude code for compilation. Unlike the #if directive, however, they test only for the existence of an identifier, not for its value.

The <condition> must be a simple logical condition; that is, a single test using one basic comparison operator (=, ==, <, >, <=, >=, <>). You may nest conditional compilation directives.

If the identifier is not defined, the <condition> always evaluates to *false*.

Conditional compilation is useful when maintaining different versions of the same program , for debugging, and for managing the use of #include files. Using #if for conditional compilation is different than conditionally executing code with an IF statement. With IF, the code still gets compiled into the resulting byte code file, even if it is never executed. By using #if to exclude code you don't want for a particular version of your program, the code is never compiled into byte code.

When *dBASE Plus*'s preprocessor processes a file, it internally defines the preprocessor identifier __version__ (two underscores on both ends) with the current version number. Earlier versions of dBASE used __dbasewin__ and __vdb__. Use these three built-in identifiers to manage code that's intended to run on different versions of dBASE.

The numeric values returned by these identifiers are as follows:

__**dbasewin**__ **:** 5.0 for dBASE 5.0, and 5.5, 5.6 or 5.7 for the versions of Visual dBASE 5.x

__**vdb**__ **:** 7.0, 7.01, 7.5 for Visual dBASE 7.x and 2000 for dBASE versions after Visual dBASE 7.5

__**version**__ **:** 1.0, 2.0, etc. depending on the release of a dBASE version after Visual dBASE 7.5

Note

The display of the above values will be affected by the number of decimal places specified by SET DECIMALS, and the separator specified by SET POINT.

#ifdef

Topic group Related topics Example

Controls conditional compilation of code based on the existence of an identifier created with #define.

Syntax

```
#ifdef <identifier>
<statements 1>
[#else
<statements 2>]
#endif
```

<identifier>

The identifier you want to test for. <identifier> is defined with the #define directive.

<statements 1>

Any number of statements and preprocessor directives. These lines are compiled if <identifier> has been defined.

#else <statements 2>

Specifies the lines to compile if <identifier> has not been defined.

Description

Use the #ifdef directive to conditionally compile sections of source code. If you've defined <identifier> with #define, the code you specify with <statements 1> is compiled; otherwise, the code following #else, if any, is compiled.

You may nest conditional compilation directives.

Conditional compilation is useful when maintaining different versions of the same program, for debugging purposes, and for managing the use of #include files. Using #ifdef for conditional compilation is different than not executing code with an IF statement. With IF, the code still gets compiled into the resulting byte code file, even if it is never executed. By using #ifdef to exclude code you don't want for a particular version of your program, the code is never compiled into byte code.

#ifndef

Topic group Related topics Example

Controls conditional compilation of code based on the existence of an identifier assigned with #define.

Syntax

#ifndef <identifier>
<statements 1>

[#else
<statements 2>]

#endif

<identifier>

The identifier you want to test for. <identifier> is defined with the #define directive.

<statements 1>

Any number of statements and preprocessor directives. These lines are compiled if <identifier> has not been defined.

#else <statements 2>

Specifies the lines to compile if <identifier> has been defined.

Description

Use the #ifndef directive to conditionally compile sections of source code. If you haven't defined <identifier> with #define, the code you specify with <statements 1> is compiled; otherwise, the code following #else, if any, is compiled.

Use #ifndef if you want to include code only if the identifier is not defined. Otherwise, you can use #ifdef to include code only if the identifier is defined, and #ifdef with its #else option to include different sets of code depending on the existence of the identifier.

You may nest conditional compilation directives.

Conditional compilation is useful when maintaining different versions of the same program, for debugging purposes, and for managing the use of #include files. Using #ifndef for conditional compilation is different than not executing code with an IF statement. With IF, the code still gets compiled into the resulting byte code file, even if it is never executed. By using #ifndef to

exclude code you don't want for a particular version of your program, the code is never compiled into byte code.

#include

Topic group Related topics Example

Inserts the contents of the specified source file (known as an include or header file) into the current program file at the location of the #include statement.

Syntax

#include <filename> | "<filename>" | :<sourceAlias>:filename

<filename> | "<filename>" | :<sourceAlias>:filename

The name of the file, optionally including a full, partial path or source alias, whose contents are to be inserted into the current program file. You can specify the file name within or without quotes. An include file typically has an .h file-name extension.

The preprocessor uses the following search order:

1. Check for a specified Source Alias. Substitute a path for the source alias and check if the file exists
2. If no source alias, or file not found, Search for the file exactly as specified in the #include statement. If a path is included, the path specified will be searched.
3. If no path is included, the current directory will be searched.
4. If not found, check if a file extension was specified. If not, add ".h" to the filename and search again for the file in the current folder or specified path.
5. If not found and no path was specified and _app.useUACPaths is true:
   search for file in
   <CSIDL_LOCAL_APPDATA>\<dBASE Plus subpath>\include
   If not found search for file in:
   <CSIDL_COMMON_APPDATA>\<dBASE Plus subpath>\include
6. If not found (or _app.useUACPaths is false) search for file in:
   _dbwinhome + "\include"
7. If not found, search for the file in path specified via INCLUDE environment variable.
   (Note that INCLUDE must contain only one path with a trailing backslash in order for a search to be successful.)

Description

The effect of #include is as if the contents of the specified file were typed into the current program file at the location of the #include statement. The specified file is called an include file. #include is used primarily for files which have #define directives.

Identifiers are available only to the program in which they are defined. To use a single set of identifiers in multiple programs, save the #define statements in a file, then use the #include directive to define the identifiers in additional programs.

An advantage of having all the #define statements in one file is the ease of maintenance. If you need to modify any of the #define statements, you need only change the include file; the program files that use the #define statements remain unchanged. After you modify the include file, recompile your program file for the changes to take effect.

To use #include directives in WFM and REP files generated by the Form and Report designers, place the directives in the Header section of the file so that the definitions will not be erased by the designer.

#pragma

Related topics Example

Sets compiler options.

Syntax

#pragma <compiler option>

<compiler option>

The compiler option to set.

Description

Use the #pragma to set compiler options. The only option supported in this version of *dBASE Plus* is:

coverage(ON | OFF)

Enables or disables the inclusion of coverage analysis information in the resulting byte code file.

Coverage analysis provides information about which program lines are executed. To provide coverage analysis, a program file must be compiled to include the extra coverage analysis information.

SET COVERAGE controls whether programs are compiled with coverage information. Use #pragma coverage( ) in your program file to override the SET COVERAGE setting for that particular file.

Note

Do not specify both #pragma coverage(ON) and #pragma coverage(OFF) in the same program file. The last #pragma takes effect; all others are ignored.

For more information about coverage files, see SET COVERAGE.

#undef

Topic group Related topics Example

Removes the current definition of the specified identifier previously defined with #define.

Syntax

#undef <identifier>

<identifier>

The identifier whose definition you want to remove.

Description

The #undef directive removes the definition of an identifier previously defined with the #define directive. If you use #define with <replacement text>, the preprocessor replaces all instances of the identifier with the replacement text from the point it encounters that #define until it encounters an #undef specifying the same identifier. Therefore, to replace an identifier only in parts of a program, insert #undef <identifier> into your program where you want the search-and-replace process to stop.

#undef is also required if you want to change the <replacement text> for an identifier. You cannot use #define for an identifier that is already defined. You must #undef the identifier first, then specify a new #define directive.

Attempting to #undef an identifier that is not defined has no effect; no error is generated.

Preprocessor Identifiers

Topic group Related topics Example

Identifies the current dBASE or *dBASE Plus* version number.

Description

When *dBASE Plus*'s preprocessor processes a file, it internally defines the preprocessor identifier __version__ (two underscores on both ends) with the current version number. Earlier versions of dBASE used __dbasewin__ and __vdb__. Use these three built-in identifiers to manage code that's intended to run on different versions of dBASE.

The numeric values returned by these preprocessor identifiers are as follows:

__**dbasewin**__ **:** 5.0 for dBASE 5.0, and 5.5, 5.6 or 5.7 for the versions of Visual dBASE 5.x

__**vdb**__ **:** 7.0, 7.01, 7.5 for Visual dBASE 7.x and 2000 for *dBASE Plus*

__**version**__ **:** 0.1, 0.2, 0.3, 0.4, etc. depending on the release of *dBASE Plus*

Note

The display of the above values will be affected by the number of decimal places specified by SET DECIMALS, and the separator specified by SET POINT.

Preprocessor Identifiers

Topic group Related topics Example

Identifies the current dBASE or *dBASE Plus* version number.

Description

When *dBASE Plus*'s preprocessor processes a file, it internally defines the preprocessor identifier __version__ (two underscores on both ends) with the current version number. Earlier versions of dBASE used __dbasewin__ and __vdb__. Use these three built-in identifiers to manage code that's intended to run on different versions of dBASE.

The numeric values returned by these preprocessor identifiers are as follows:

__**dbasewin**__ **:** 5.0 for dBASE 5.0, and 5.5, 5.6 or 5.7 for the versions of Visual dBASE 5.x

__**vdb**__ **:** 7.0, 7.01, 7.5 for Visual dBASE 7.x and 2000 for *dBASE Plus*

__**version**__ **:** 0.1, 0.2, 0.3, 0.4, etc. depending on the release of *dBASE Plus*

Note

The display of the above values will be affected by the number of decimal places specified by SET DECIMALS, and the separator specified by SET POINT.

Preprocessor Identifiers

Topic group Related topics Example

Identifies the current dBASE or *dBASE Plus* version number.

Description

When *dBASE Plus*'s preprocessor processes a file, it internally defines the preprocessor identifier __version__ (two underscores on both ends) with the current version number. Earlier versions of dBASE used __dbasewin__ and __vdb__. Use these three built-in identifiers to manage code that's intended to run on different versions of dBASE.

The numeric values returned by these preprocessor identifiers are as follows:

__**dbasewin**__ **:** 5.0 for dBASE 5.0, and 5.5, 5.6 or 5.7 for the versions of Visual dBASE 5.x

__**vdb**__ **:** 7.0, 7.01, 7.5 for Visual dBASE 7.x and 2000 for *dBASE Plus*

__**version**__ **:** 0.1, 0.2, 0.3, 0.4, etc. depending on the release of *dBASE Plus*

Note

The display of the above values will be affected by the number of decimal places specified by SET DECIMALS, and the separator specified by SET POINT.

BDE Limits

BDE Limits

The following tables list the Borland Database Engine and Native dBASE /Paradox File Maximum Limits for both 16 and 32 bit Versions of BDE. If you find you cannot reach these limits, or are getting an out of memory error, increasing your SHAREDMEMSIZE in BDE Config to 4096 or more should allow you to reach these limits.

GENERAL LIMITS

Description	Limit
Clients in system	48
Sessions per client (3.5 and earlier, 16 Bit, 32 Bit)	32
Session per client (4.0, 32 Bit)	256
Open databases per session (3.5 and earlier, 16 Bit, 32 Bit)	32
Open databases per session (4.0, 32 Bit)	2048
Loaded drivers	32
Sessions in system (3.5 and earlier, 16 Bit, 32 Bit)	64
Sessions in system (4.0, 32 Bit)	12288
Cursors per session	4000
Entries in error stack	16
Table types per driver	8
Field types per driver	16
Index types per driver	8
Size of configuration (IDAPI.CFG) file	48K
Size of SQL statement (RequestLive=False)	64K
Size of SQL statement (RequestLive=True)	4K
Size of SQL statement (RequestLive=True) (4.01, 32 Bit)	6K
Record buffer size (SQL or ODBC)	16K
Table and field name size in characters	31
Stored procedure name size in characters	64
Fields in a key	16
File extension size in characters	3
Table name length in characters (some servers might have other limits)	260

Path and file name length in characters	260

DBASE Limits

Description	Limit
Open dBASE tables per system (16 Bit)	256
Open dBASE tables per system (BDE 3.0 - 4.0, 32 Bit)	350
Open dBASE tables per system (BDE 4.01, 32 Bit)	512
Record locks on one dBASE table (16 and 32 Bit)	100
Records in transactions on a dBASE table (32 Bit)	100
Records in a table	1 Billion
Bytes in .DBF (Table) file	2 Billion
Size in bytes per record (dBASE 4)	4000
Size in bytes per record (dBASE for Windows)	32767
Number of fields per table (dBASE 4)	255
Number of fields per table (dBASE for Windows)	1024
Number of index tags per .MDX file	47
Size of character fields	254
Open master indexes (.MDX) per table	10
Key expression length in characters	220

Paradox Limits

Description	Limit
Tables open per system (4.0 and earlier, 16 Bit, 32 Bit)	127
Tables open per system (4.01, 32 Bit)	254
Record locks on one table (16Bit) per session	64
Record locks on one table (32Bit) per session	255
Records in transactions on a table (32 Bit)	255
Open physical files (4.0 and earlier, 16 Bit, 32 Bit) (DB, PX, MB, X??, Y??, VAL, TV)	512
1024 Open physical files (4.01, 32 Bit) (DB, PX, MB, X??, Y??, VAL, TV)	1024
Users in one PDOXUSRS.NET file	300
Number of fields per table	255
Size of character fields	255
Records in a table	2 Billion
Bytes in .DB (Table) file	2 Billion
Bytes per record for indexed tables	10800
Bytes per record for non-indexed tables	32750
Number of secondary indexes per table	127
Number of fields in an index	16
Concurrent users per table	255
Megabytes of data per BLOb field	256

Passwords per session	100
Password length	15
Passwords per table	63
Fields with validity checks (32 Bit)	159
Fields with validity checks (16 Bit)	63

Index

邓
- 23, 24
-- 24
'
' symbols 30
!
! 523
#
25
symbol 33
#define 883
#else 886
#endif 886
#if 886
#ifdef 887
#ifndef 888
#include 889
#pragma 889
#undef 890
$
$ 25
%
%24
%= 22
&
& 29
&& symbol 31
(
() 28
*
* 24
* symbol 31
** 24
*/ symbol 31
*=22
.
. (period) 26
.AND. 24
.NOT. 24
.OR. 24
/
/ 24
/* symbol 31
// symbol 31
/=22
?
? 793
?? 796
??? 796
[
[] 26
[] symbols 30
^
^ 24
_
__dbasewin__ 891
__vdb__ 891
__version__ 891
_alignment 806
_app 62
_app.frameWin 64
_dbaselock field 171
_dbwinhome 556
_indent 807
_lmargin 807
_padvance 808
_pageno 809
_pbpage 809
_pcolno 810
_pcopies 810
_pdriver 811
_pecode 811
_peject 812
_pepage 812
_pform 813
_plength 813
_plineno 814
_ploffset 814
_porientation 815
_ppitch 815
_pquality 816
_pscode 817
_pspacing 817
_rmargin 817
_tabs 818
_wrap 819
+
+ 22, 24
++ 24
+= 22
<
< 25
< > notation 18
<= 25
<> 25
=
= 22, 25
-=22
== 25
>
> 25
->28
>= 25
2
2000 733
and SET EPOCH 733
A
abandon() 185
abandonRecord() 335

abandonUpdates() 187
ABS() 748
absolute value 748
accelerators 466
access() 187, 846
accessDate() 523
ACOPY() 677
ACOS() 748
active 187
activeControl 336
activeX
djvClockX 833
ActiveX class 275, 819
add() 677
addPassword() 188
addToMRU( ) method 73
ADEL() 672, 678
ADIR() 672, 678
ADIREXT() 672, 679
advise() 866
AELEMENT() 672, 679
aelement_func 679
AFIELDS() 672, 680
AFILL() 672, 680
agAverage() 489
agCount() 490
aggregate functions 662
agMax() 491
agMin() 491
AGROW() 672, 681
agStdDeviation() 492
agSum() 492
agVariance() 493
AINS() 672, 681
ALEN() 682
alerts 94
alias 336
alias operator 28
ALIAS() 559
aliases 659
table 659
alignHorizontal 336
alignment 337, 806
alignVertical 338
allowAddRows 339
allowColumnMoving 339
allowColumnSizing 339
allowDEOExeOverride 74
allowDrop 339
allowEditing 340
allowEditLabels 340
allowEditTree 340
allowFullScrollRange 341
allowRowSizing 341
allowYieldOnMsg 75
alphabetic 773
first character in string 773
ALTER TABLE 665
ALTERNATE 797, 802
CLOSE 797
SET 802
alwaysDrawCheckBox 341
anchor 342
AND 24
annotating code 31
ANSI() 847
app 62
app.frameWin 64
APPEND AUTOMEM 560
APPEND command 560
APPEND FROM 561
APPEND FROM ARRAY 563
APPEND MEMO 564
append property 342
append() 189
appendUpdate() 189
Application shell 62
applyFilter() 190
applyLocate() 190
applyUpdates() 191
appshell 62
appSpeedBar 343
arccosine 748
arcsine 749
arctangent 749, 750
ARESIZE() 672, 683
ARGCOUNT() 119
ARGVECTOR() 120
arithmetic operators 22, 23
Array 581, 621, 672, 673, 676
COPY TO 581
REPLACE FROM 621
Array class 673
Array functions 672
array symbol 32
arrays 9
ASC() function 768
asc() method 768
ASCAN() 672, 683
ASIN() 749
ASORT() 672, 684
assigning variables 8
Assignment operators 22
AssocArray class 676
ASUBSCRIPT() 672, 684
AT() 768
ATAN() 749
atFirst() 192
atLast() 192
ATN2() 750
attach() method 76
attributes 1

language ... 1
autoCenter ... 343
autoDrop ... 344
autoEdit ... 193
autoLockChildRows ... 193
AUTOMEM ... 560, 572, 617, 620, 648
APPEND ... 560
CLEAR ... 572
RELEASE ... 617
REPLACE ... 620
STORE ... 648
AUTONULLFIELDS ... 54, 194
autoNullFields property ... 54, 194
SET AUTONULLFIELDS ... 54
AUTOSAVE ... 630
SET ... 630
autoSize ... 344
autoSort ... 493
autoTrim ... 344
average ... 564
AVG() ... 662
B
background ... 345
Band class ... 482
base 10 logarithm ... 755
baseClassName property ... 120
Basic caching ... 269
behavior ... 269
pros and cons ... 269
basic expressions ... 7
basic language attributes ... 1
basic statements ... 10
BDE ... 658
BDE alias ... 160
BDE Limits ... 892
before ... 345
beforeCellPaint ... 346
beforeCloseUp ... 346
beforeDropDown ... 348
beforeEditPaint ... 348
beforeGetValue ... 194
beforeRelease ... 347
beginAppend() ... 195, 348
beginEdit() ... 196
beginFilter() ... 197
beginLocate() ... 197
beginNewFrame ... 494
beginNewFrame() ... 494
BEGINTRANS() ... 198, 565
beginTrans() method ... 198
BELL ... 55
SET ... 55
bgColor ... 349
BINARY ... 577, 620
COPY ... 577
REPLACE ... 620
binary and OLE data ... 5
BINTYPE() ... 566
BITAND() ... 842
BITLSHIFT() ... 842
bitmapAlignment ... 350
BITNOT() ... 843
BITOR() ... 843
BITRSHIFT() ... 843
BITSET() ... 844
Bitwise ... 841
BITXOR() ... 844
BITZRSHIFT() ... 845
BLANK ... 567
block comment symbol ... 31
BLOCKSIZE ... 55
SET ... 55
BOF() ... 568
bold ... 350
BOOKMARK() ... 199, 568
bookmarksEqual() ... 199
border ... 351
borders ... 452
scrolling a client area ... 452
borderStyle ... 351
Borland Database Engine (BDE) ... 658
bottom ... 352
braces (symbol) ... 32
BROWSE ... 569
Browse class ... 276
BUILD ... 33
buttons ... 352
C
cacheUpdates ... 200
Caching ... 269
basic ... 269
dynamic ... 269
CALCULATE ... 570
calculated fields ... 168, 442
and picture templates ... 442
creating ... 168
call operator ... 28
CALLBACK ... 867
GETCALLADDRESS ... 871
RELEASECALLBACK ... 879
canAbandon ... 200
canAppend ... 201
CANCEL ... 848
canChange ... 201, 352
canClose ... 202, 353
canDelete ... 202
canEdit ... 203
canEditLabel ... 353
canExpand ... 353
canGetRow ... 203
canNavigate ... 204, 354
canOpen ... 204

canRender 494
canSave 205
canSelChange 354
capitalization guidelines 19
CASE 120, 127
  DO 127
CATCH 121
CD 523
CDOW() 716
CEILING() 750
cellHeight 355
CENTER() 769
centering strings 769
CENTURY 731, 733
  SET 731
CERROR() 848
CGI 506
  report output 506
CHANGE() 572
changedTableName 205
characteristics 1
  language 1
characters 89
  and fonts 89
charAt() 770
charSet property 77
CHARSET() function 849
CheckBox class 278
checkBoxes 355
checked 77, 355
checkedBitmap 78
checkedImage 356
choosePrinter( ) 495
choosePrinter() 507, 796
CHR() function 770
chr() method 771
CLASS 121
class ActiveX 275, 819
class Array 673
class AssocArray 676
class Band 482
class Browse 276
class CheckBox 278
class ColumnCheckBox 279
class ColumnComboBox 280
class ColumnEditor 281
class ColumnEntryfield 283
class ColumnHeadingControl 284
class ColumnSpinBox 285
class ComboBox 286
class Container 288
class Database 161
class DataModRef 165
class DataModule 164
class Date 713
class DbError 166
class DbException 166
class DbfField 167
class DBFIndex 169
class DDELink 863
class DDETopic 864
class Designer 117
class Editor 290
class Entryfield 293
class Exception 118
class Field 168
class File 521
class Form 295
class Grid 298
class GridColumn 302
class Group 484
class hierarchy 160
  data access objects 160
class Image 303
class Index 171
class Line 304
class ListBox 306
class LockField 171
class Menu 65
class MenuBar 67
class NoteBook 307
class Number 741
class Object 119
class OLE 309
class OleAutoClient 866
class PageTemplate 485
class PaintBox 311
class Parameter 172
class PdxField 173
class Popup 68
class Progress 312
class PushButton 313
class Query 174
class RadioButton 315
class Rectangle 316
class Report 486
class ReportViewer 317
class Rowset 175
class ScrollBar 319
class Session 180, 261
class Shape 320
class Slider 321
class SpinBox 322
class SqlField 181
class StoredProc 182
class StreamFrame 487
class StreamSource 489
class String 766
class SubForm 324
class TabBox 327
class TableDef 183
class Text 328

class TextLabel 330
class Timer 715
class ToolBar 69
class ToolButton 72
class TreeItem 331
class TreeView 333
class UpdateSet 184
classes 9, 13, 14, 15, 16
  and execution 16
  and objects 13, 14, 15
  array 9
  example 15
classId 356
className 122
CLEAR 797
CLEAR ALL 35
CLEAR AUTOMEM 572
CLEAR FIELDS 573
CLEAR MEMORY 123
CLEAR PROGRAM 123
CLEAR TYPEAHEAD 78
clearFilter() 206
clearRange() 206
clearTics() 356
clientEdge 357
close 36
CLOSE ALL 36
CLOSE ALTERNATE 797
CLOSE DATABASES 573
close forms 36
CLOSE INDEXES 574
CLOSE PRINTER 797
CLOSE PROCEDURE 124
CLOSE TABLES 574
close() 206, 357, 524
closing dBASE Plus 144
CMD 1
CMONTH() 717
code 118
code signing 33
codeblock symbol 32
codeblocks and functions 11, 12, 13
codePage 207
colon (symbol) 31
color 80, 507
  DEFINE 80
Color Property Builder 358, 359, 364, 365
colorColumnLines 358
colorHighlight 358
colorNormal 359
colorRowHeader 364
colorRowLines 364
colorRowSelect 365
colors 80, 88
  custom 80
ColumnCheckBox class 279
ColumnComboBox class 280
columnCount 365
ColumnEditor class 281
ColumnEntryfield class 283
ColumnHeadingControl class 284
columnNo 365
columns 366, 385, 386, 393, 394, 395, 659
  Grid heading fonts and color 393, 394, 395
  Grid text fonts 385, 386
  naming (local SQL) 659
ColumnSpinBox class 285
ComboBox class 286
COMMAND 1, 39
  CREATE 39
Command Line Switches 1
comment symbols 31
comments 16
COMMIT() 207, 574
common component properties 273
common control properties 273
compare numbers 755
  maximum 755
  minimum 755
comparison operators 25
COMPILE 37
complex expressions 10
component properties 273
  common 273
CONFIRM 101
  SET 101
CONSOLE 803
  SET 803
constrained 207
constraints 664
  local SQL 664
Container class 288
context 166, 496
contextHelp 366
continuation symbol 32
CONTINUE 575
control statements 11
controls 273
  common properties 273
conventions 6, 17, 659
  names 6
  naming 659
  syntax 17
CONVERT 38
converting 8, 9, 22, 713, 717, 718, 720, 721, 722, 723, 738, 739, 740, 748, 750, 751, 752, 754, 760, 768, 770, 771, 780, 785, 786, 790, 792
  all types to string 22
  ASCII value to character 770, 771
  automatic type conversion 8
  character to ASCII value 768
  date to string 720, 721, 738, 739

datetime to date ... 722
datetime to string ... 722
datetime to time ... 723
degrees to radians ... 751
explicit type conversion ... 9
number to absolute value ... 748
number to integer ... 750, 752, 754, 760
number to string ... 786
radians to degrees ... 760
string to date ... 713, 717
string to datetime ... 718
string to lowercase ... 780, 790
string to number ... 792
string to proper case ... 780, 790
string to sound-alike ... 785
string to time ... 718
string to uppercase ... 790, 792
time to string ... 740
type conversion ... 8
copies ... 507
COPY ... 576
COPY BINARY ... 577
COPY FILE ... 525
COPY MEMO ... 578
COPY STRUCTURE ... 579
COPY STRUCTURE EXTENDED ... 580
COPY TABLE ... 581
COPY TO ARRAY ... 581
copy() ... 208, 366, 524
copying data ... 671
local SQL ... 671
copying records ... 671
local SQL ... 671
copyTable() ... 208
copyToFile() ... 209
COS() ... 751
cosine ... 751
count ... 582
count() ... 209, 367, 662, 685
COVERAGE ... 46, 56
DISPLAY ... 46
SET ... 56
CREATE ... 39
CREATE COMMAND ... 39
CREATE DATAMODULE ... 40
CREATE FILE ... 41
CREATE FORM ... 41
CREATE INDEX ... 665
CREATE LABEL ... 42
CREATE MENU ... 43
CREATE POPUP ... 43
CREATE PROJECT ... 44
CREATE QUERY ... 44
CREATE REPORT ... 45
CREATE SESSION ... 583
CREATE TABLE ... 666
create() ... 526
CREATE...FROM ... 584
CREATE...STRUCTURE EXTENDED ... 585
createDate() ... 527
createIndex() ... 210
createTime() ... 527
CTOD() ... 717
CTODT() ... 718
CTOT() ... 718
CUAENTER ... 101
SET ... 101
cuaTab ... 367
CURRENCY ... 761
SET ... 761
currentColumn ... 368
currentUserPath ... 79
curSel ... 368
CUSTOM ... 121
custom colors ... 80
cut() ... 368

D

data ... 668, 669, 670, 671, 672
copying (local SQL) ... 671
modifying (local SQL) ... 672
retrieving (local SQL) ... 668, 669, 670, 671
data definition ... 661
local SQL ... 661
data manipulation ... 662
local SQL ... 662
Data objects ... 160
data type mappings ... 666
local SQL ... 666
data types ... 2, 3, 4, 5
binary and OLE ... 5
date ... 4
logical ... 3
memo ... 5
null value ... 4
numeric ... 3
OLE and binary ... 5
string ... 2
DATABASE ... 210, 612, 630
OPEN ... 612
SET ... 630
Database class ... 161
DATABASE() ... 586
databaseName ... 211
databases ... 80, 161, 573
accessing ... 161
CLOSE ... 573
property ... 80
database-specific data types ... 4
dataLink ... 369
dataModClass ... 211
DataModRef class ... 165
DATAMODULE ... 40

CREATE ... 40
DataModule class ... 164
dataSource ... 369, 371
date ... 712, 718, 732, 738, 739
convert to GMT ... 738
convert to local ... 739
convert to standard ... 739
current ... 718
SET ... 732
Date class ... 713
date data ... 4
date function ... 663
local SQL ... 663
DATE TO ... 733
SET ... 733
DATE() ... 528, 718
DATETIME() ... 719
day of month ... 719, 724, 735
set Date object ... 735
day of week ... 720, 725
DAY() ... 719
dBASE Plus ... 1
language definition ... 1
DBASE_SUPPRESS_STARTUP_DIALOGS 849
DbError class ... 166
DBERROR() ... 850
DbException class ... 166
DBF() ... 586
DbfField class ... 167
DBFIndex class ... 169
DBMESSAGE() ... 850
DBTYPE ... 631
SET ... 631
dbwinhome ... 556
DDE ... 863
DDE server ... 880
DDELink class ... 863
ddeServiceName ... 80
DDETopic class ... 864
DEBUG ... 45
debugger ... 45
debugging ... 846
decimalLength ... 211
decimals ... 761
SET ... 761
DECLARE ... 685
decrement operators ... 24
default ... 212, 371
DEFINE ... 125
DEFINE COLOR ... 80
degrees ... 760
from radians ... 760
DELETE ... 587, 667
DELETE FILE ... 529
DELETE TABLE ... 588
DELETE TAG ... 588
delete() ... 212, 528, 686
DELETED ... 631
SET ... 631
DELETED() ... 589
deleting indexes ... 667
local SQL ... 667
deleting records ... 667
local SQL ... 667
deleting tables ... 668
local SQL ... 668
delimiters ... 31
DEO - Dynamic External Objects ... 517
descending ... 213
DESCENDING() ... 589
description ... 372
DESIGN ... 57
SET ... 57
Designer class ... 117
designView ... 372
destination ... 213
detach() ... 81
detailBand ... 496
detailNavigationOverride ... 82
DEVELOPMENT ... 57
SET ... 57
difference ... 771
DIFFERENCE() ... 771
dimensions ... 688
DIR ... 529
dir() ... 689
directive ... 33
preprocessor symbol ... 33
directives ... 17
preprocessor definition ... 17
DIRECTORY ... 551
SET ... 551
dirExt() ... 690
disabledBitmap ... 373
disablePopup ... 373
DISKSPACE() ... 530
DISPLAY ... 590
DISPLAY COVERAGE ... 46
DISPLAY FILES ... 531
DISPLAY MEMORY ... 47
DISPLAY STATUS ... 48
DISPLAY STRUCTURE ... 49
djvClockX ... 833
DLL ... 872, 878
LOAD ... 872
RELEASE ... 878
DLLs ... 863
DML statements ... 662
local SQL ... 662
DMY() ... 719
DO ... 126
DO CASE ... 127

DO WHILE 129
DO...UNTIL 129
DOS 531
double-byte values 442
double-quotation mark symbols 30
doVerb() 373
DOW() 720
downBitmap 374
Drag&Drop 339, 374, 375, 420, 421, 422, 423
  allowDrop 339
  and Windows applications 421, 422, 423
  drag() 374
  dragEffect 375
  from dBASE Plus Navigator 423
  onDragBegin 420
  onDragEnter 421
  onDragLeave 421
  onDragOver 422
  onDrop 423
drag() 374
dragEffect 375
dragScrollRate 376
drawMode 376
drillDown 496
driverName 213
DROP INDEX 667
Drop Source objects 275, 276, 278, 286, 288, 290, 293, 298, 303, 304, 306, 307, 309, 311, 312, 313, 315, 316, 317, 319, 320, 321, 322, 327, 328, 330, 331, 333, 819
DROP TABLE 668
Drop Target objects 276, 288, 295, 298, 303, 306, 307, 311, 317, 324, 333
dropDownHeight 377
dropDownWidth 377
dropIndex() 214
dropTable() 214
DTOC() 720
DTODT() 721
DTOR() 751
DTOS() 721
DTTOC() 722
DTTOD() 722
DTTOT() 723
duplex 507
Dynamic caching 269
  behavior 269
  conditions 269
  pros and cons 269
Dynamic External Objects 517
dynamic subclassing 14

E

EDIT 591
editCopyMenu 82
editCutMenu 83
editor 59
  SET 59
Editor class 290
editorControl 377
editorType 378
editPasteMenu 83
editUndoMenu 84
EJECT 798
EJECTPAGE 798
ELAPSED() 723
element() 692
elements 378, 557, 558
  Xbase command 557, 558
ELSE 130
ELSEIF 130
EMPTY() 130
emptyTable() 215
enabled 378, 724
enableSelection 379
enclosing strings 30
ENCRYPTION 858
  SET 858
ENDFOR 133
  FOR 133
endOfSet 215
endPage 495, 497
ENDPRINTJOB 800
endSelection 379
ensureVisible() 379
Entryfield class 293
ENUMERATE() 131
eof() 531, 591
EPOCH 733
  SET 733
ERASE 532
ERROR 854, 858
  ON 854
  SET 858
error() 532, 850
errorAction 84
errorHTMFile 85
errorLogFile 86
errorLogMaxSize 86
errors 846
errorTrapFilter 87
ESCAPE 96, 102
  ON 96
  SET 102
escExit 380
evalTags 380
EXACT 632
  SET 632
exactMatch 216
example 17
  program 17
Exception class 118
EXCLUSIVE 632

SET ... 632
execute() ... 216, 868
executeMessages( ) method ... 87
executeSQL() ... 216
execution ... 16
functions and classes ... 16
exeName ... 88
exists() ... 533
EXIT (loop) ... 131
exit (program) ... 144
EXP() ... 752
expandable ... 497
expanded ... 381
exponent ... 752
logarithm ... 752
expression codeblock symbol ... 32
expressions ... 7, 10, 21
basic ... 7
complex ... 10
evaluating ... 21
EXTENDED ... 580, 585
COPY STRUCTURE ... 580
CREATE...STRUCTURE ... 585
extending dBASE Plus ... 863
EXTERN ... 868
EXTRACT() ... 663

F

FACCESSDATE() ... 516
FCLOSE() ... 517
FCOUNT() ... 593
FCREATE() ... 517
FCREATEDATE() ... 516
FCREATETIME() ... 516
FDATE() ... 516
FDECIMAL() ... 592
FEOF() ... 517
FERROR() ... 517
FFLUSH() ... 517
FGETS() ... 517
Field class ... 168
field morphing ... 194, 201, 442
and picture templates ... 442
reversing ... 201
field names ... 8
using in expressions ... 8
FIELD() ... 592
fieldName ... 217
fields ... 31, 217, 381, 442, 573, 633
calculated and picture templates ... 442
CLEAR ... 573
naming with spaces in them ... 31
SET ... 633
fields() ... 692
FILE ... 41, 525, 529
COPY ... 525
CREATE ... 41
DELETE ... 529
File class ... 521
File information functions ... 516
File utility commands ... 516
FILE() ... 533
filename property ... 118, 132, 218, 382
filename skeleton ... 6
files ... 531, 540
DISPLAY ... 531
LIST ... 540
fill() ... 693
FILTER ... 218, 635
SET ... 635
filterOptions ... 219
FINALLY ... 132
FINDINSTANCE() ... 132
findKey() ... 219
findKeyNearest() ... 220
first ... 382
first() ... 221
firstChild ... 382
firstColumn ... 383
firstKey ... 694
firstOnFrame ... 497
firstPageTemplate ... 498
firstRow( ) method ... 383
firstVisibleChild ... 384
fixed ... 498
fixing bugs ... 45
FLDCOUNT() ... 593
FLDLIST() ... 593
FLENGTH() ... 594
FLOCK() ... 594
FLOOR() ... 752
FLUSH ... 595
flush() ... 222, 534
FNAMEMAX() ... 534
focus ... 384
focusBitmap ... 384
fontBold ... 385
fontItalic ... 385
fontName ... 385
fonts ... 89, 385, 386, 393, 394, 395
Grid headings ... 393, 394, 395
Grid text ... 385, 386
fontSize ... 386
fontStrikeout ... 386
fontUnderline ... 386
footerBand ... 499
FOPEN() ... 517
for ... 11
FOR() ... 596
FOR...ENDFOR ... 133
for...in ... 11
forExpression ... 222
form ... 41, 387

CREATE ... 41
Form class ... 295
formatting strings ... 791
forward slash (symbol) ... 31
FOUND() ... 596
FPUTS() ... 517
frameWin ... 64
FREAD() ... 517
FROM ... 561, 584
APPEND ... 561
CREATE ... 584
FROM ARRAY ... 563, 621
APPEND ARRAY APPEND FROM ... 563
REPLACE ... 621
FROM clause ... 669
frozenColumn ... 387
FSEEK() ... 517
FSHORTNAME() ... 516
FSIZE() ... 516
FTIME() ... 516
FULLPATH ... 551
SET ... 551
FUNCTION ... 103, 134, 388
SET ... 103
function pointers ... 12
functions ... 16, 598
allowed in DBF index ... 598
and execution ... 16
functions (local SQL) ... 662, 663
aggregate ... 662
date ... 663
string ... 663
functions and codeblocks ... 11, 12, 13
FUNIQUE() ... 535
FV() ... 753
FWRITE() ... 517

G

GENERATE ... 597
getByte() ... 772
GETCALLADDRESS ... 871
GETCOLOR() ... 88
getColumnObject() ... 389
getColumnOrder() ... 389
getDate() ... 724
getDay() ... 725
GETDIRECTORY() ... 535
GETENV() ... 535
GETFILE() ... 536
getFile() method ... 695
GETFONT() ... 89
getHours() ... 725
getItemByPos() ... 390
getMinutes() ... 726
getMonth() ... 726
gets() ... 538
getSchema() ... 222
getSeconds() ... 727
getTextExtent() ... 390
getTime() ... 727
getTimezoneOffset() ... 727
getYear() ... 728
GMT ... 738
convert date ... 738
GO ... 597
goto() ... 223
greater than ... 750
nearest integer ... 750
Grid class ... 298
Grid heading fonts and color ... 393, 394, 395
Grid text fonts ... 385, 386
GridColumn class ... 302
gridLineWidth ... 391, 499
group ... 391
GROUP BY clause ... 670
Group class ... 484
groupBy ... 499
grow() ... 697
guidelines ... 19
capitalization ... 19

H

handle ... 224, 391, 539
hasButtons ... 391
hasColumnHeadings ... 392
hasColumnLines ... 392
hasHScrollBar( ) method ... 89
hasIndicator ... 392
hasLines ... 393
hasRowLines ... 393
hasVScrollBar( ) method ... 90
hasVScrollHintText ... 393
HAVING clause ... 671
headerBand ... 500
headerEveryFrame ... 500
headingColorNormal ... 393
headingControl ... 394
headingFontBold ... 394
headingFontItalic ... 394
headingFontName ... 394
headingFontSize ... 395
headingFontStrikeout ... 395
headingFontUnderline ... 395
HEADINGS ... 636
SET ... 636
height ... 396
HELP ... 51, 880
helpFile ... 396
helpId ... 397
heterogeneous joins ... 671
hexadecimal ... 3
Hi Precision ... 747
HOME() ... 539
horizontal scroll bars ... 89

hours 725, 735
set Date object 735
hScrollBar 397
HTML 506
report output 506
HTOI() 845
hWnd 398
hWndClient 398
hWndParent 398
I
IBLOCK 60
SET 60
icon 399
id 399
ID() 851
IDE overview 33
if 11, 135
IIF() 136
image 399, 879
RESTORE 879
Image class 303
imageScaleToFont 400
imageSize 400
imgPixelHeight property 400
imgPixelWidth property 401
increment operators 24
indent 401, 807
inDesign 402
INDEX 598, 636, 665, 667
CREATE 665
DROP 667
SET 636
Index class 171
indexes 574, 665, 667
CLOSE 574
creating (local SQL) 665
deleting (local SQL) 667
indexes property 224
indexName 224, 225
indexOf() 772
initiate() 871
INKEY() 90
INSERT 668
insert() 699
inserting records 668
local SQL 668
INSPECT() 51
INT() 754
integer 750, 752, 754
from number 754
less-than 752
nearest 750
integralHeight 402
interval 728
introduction 117, 160, 273, 479, 765
core language 117
data access objects 160
Form objects 273
Report objects 479
String object 765
introduction to Xbase 557
inverse cosine 748
inverse sine 749
inverse tangent 749, 750
ISALPHA() function 773
isAlpha() method 773
ISBLANK() 602
isInherited ( ) 137
isKey() 701
isLastPage() 501
ISLOWER() function 774
isLower() method 774
isolationLevel 225
isRecordChanged() 402
isRowLocked() 225
isSetLocked() 225, 226
ISTABLE() 603
ISUPPER() function 774
isUpper() method 775
ITOH() 846
J
joins 664
live 664
joins (local SQL) 671
heterogeneous 671
K
key 97, 403
ON 97
KEY TO 637
SET 637
KEY() 603
KEYBOARD 93
keyboard() 403
KEYMATCH() 604
keyViolationTableName 226
keywords 660
local SQL 660
L
LABEL 42
CREATE 42
language 227
language [_app object] 93
language definition 1
language syntax 19
languageDriver 227
last() 227
lastIndexOf() 775
lastRow( ) method 404
LDCHECK 859
SET 859
LDCONVERT 860
SET 860

lDriver property 93
LDRIVER() function 851
leading 501
leading zeroes..... 3, 786
left 405
LEFT() function 776
left() method..... 777
leftTrim()..... 777
LEN() 777
length 228, 777, 778
  character string..... 777, 778
  numeric value 778
LENNUM( ) 778
less-than 752
level 405
LIBRARY..... 149
  SET 149
LIKE 663
LIKE()..... 779
Line class..... 304
line comment symbol 31
line continuation symbol..... 32
lineNo..... 118, 405
LINENO() 852
linesAtRoot..... 405
linkFileName 406
LIST 605
LIST COVERAGE 51
LIST FILES 540
LIST MEMORY 51
LIST STATUS 51
LIST STRUCTURE 51
ListBox class..... 306
literal array symbol..... 32
literal arrays 9
literal date symbol..... 32
literal string symbols 30
literals..... 2, 3, 4
  boolean..... 3
  date 4
  null..... 4
  numeric..... 3
  string..... 2
live 228
live joins..... 664
live queries..... 663
LKSYS()..... 606
lmargin..... 807
LOAD DLL..... 872
loadChildren() 406
LOCAL 137
local SQL 658, 659, 660, 661, 662, 663, 664, 666, 671, 672
  aggregate functions..... 662
  constraints..... 664
  data definition..... 661
  data manipulation..... 662
  data type mappings..... 666
  date function..... 663
  DML statements 662
  heterogeneous joins..... 671
  live joins..... 664
  live queries 663
  naming conventions 659
  operators 659
  overview..... 658
  parameter substitutions 662
  reserved words..... 660
  restrictions..... 663, 664
  statements..... 664
  string functions 663
  updateable queries..... 663, 664
  updating data..... 672
local timezone..... 727
locale 846
LOCATE..... 607
locateNext()..... 228
locateOptions..... 229
LOCK 229, 638
  SET 638
LOCK()..... 608
lockedColumns 406
LockField class 171
lockRetryCount 230
lockRetryInterval 230
lockRow()..... 231
lockSet()..... 231
locktype..... 232
LOG()..... 754
LOG10()..... 755
logarithm 752, 754, 755
  base 10..... 755
  exponent..... 752
logical data..... 3
logical operators..... 24
logicalSubType 232
logicalType..... 233
login()..... 234
loginDBAlias property 235
loginString..... 235
LOGOUT..... 852
LOOKUP()..... 608
lookupRowset 236
lookupSQL 236
lookupTable 237
lookupType 237
LOOP..... 138
LOWER() 663, 780
lowercase..... 774, 780, 790
  convert from uppercase..... 780, 790
  first alphabetic character in string..... 774
Low-level file functions..... 517

LTRIM() ... 780
LUPDATE() ... 609
M
macro operator ... 29
manifest file ... 466
mappings ... 666
  data type (local SQL) ... 666
MARGIN ... 803
  SET ... 803
marginBottom ... 502
marginHorizontal ... 502
marginLeft ... 502
marginRight ... 503
marginTop ... 503
marginVertical ... 504
MARK ... 734
  SET ... 734
masterChild ... 238
masterFields ... 238
masterRowset ... 239
masterSource ... 239
Math
  Hi Precision ... 747
Math / Money ... 747
MAX() ... 662, 755
maximize ... 407
maximum ... 240, 755
  compare numbers to find ... 755
maxLength ... 407
maxRows ... 504
MBLOCK ... 61
  SET ... 61
MD ... 540
mdi ... 407
MDX() ... 609
MDY() ... 729
MEMLINES() ... 610
MEMO ... 564, 578, 622
  APPEND ... 564
  COPY ... 578
  REPLACE ... 622
memo data ... 5
memoEditor ... 408
MEMORY ... 47, 123
  CLEAR ... 123
  DISPLAY ... 47
MEMORY() ... 852
MEMOWIDTH ... 639
  SET ... 639
MENU ... 43
  CREATE ... 43
Menu class ... 65
MenuBar class ... 67
menuFile ... 409
message ... 104, 118
  SET ... 104
MESSAGE() ... 853
messages ... 94
methods ... 14
metric ... 409
millenium ... 733
  and SET EPOCH ... 733
milliseconds ... 730, 740
  returning ... 740
MIN() ... 662, 755
minimize ... 410
minimum ... 240, 755
  compare numbers to find ... 755
minus operator ... 23
minutes ... 726, 736
  set Date object ... 736
MKDIR ... 540
MLINE() ... 610
MOD() ... 756
mode ... 264
modified ... 240
modify ... 52, 410
MODIFY COMMAND ... 52
MODIFY DATAMODULE ... 52
MODIFY FILE ... 52
MODIFY FORM ... 52
MODIFY LABEL ... 52
MODIFY MENU ... 52
MODIFY POPUP ... 52
MODIFY PROJECT ... 52
MODIFY QUERY ... 52
MODIFY REPORT ... 52
MODIFY STRUCTURE ... 53
modifying data ... 672
  local SQL ... 672
month ... 726, 736
  set Date object ... 736
MONTH() ... 729
morphing ... 442
  and picture templates ... 442
mousePointer ... 410
move() ... 411
moveable ... 412
MSGBOX() ... 94
multiple ... 412
multiSelect ... 413
N
name ... 241, 413
Name/database delimiters ... 31
naming conventions ... 659
  local SQL ... 659
nativeCode ... 166
nativeObject ... 414
navigateByMaster ... 241
navigateMaster ... 243
NDX() ... 611
NEAR ... 640

SET ... 640
NETERROR ... 855
ON ... 855
NETWORK() ... 853
new ... 26
next() ... 243
nextKey() ... 95, 702
nextObj ... 414
nextPageTemplate ... 504
nextSibling ... 414
non-operational symbols ... 30, 31, 32, 33
noOfChildren ... 414
NOT ... 24
notation ... 18
syntax ... 18
NoteBook class ... 307
notes ... 16
adding to code ... 16
notify() ... 873
notifyControls ... 244
null values ... 4
Number ... 741
number sign (#) ... 33
NumberException ... 746
numbers ... 792
finding in strings ... 792
numeric data ... 3
numeric operators ... 22, 24
O
OBJECT ... 147, 713
moment in time ... 713
RELEASE ... 147
Object class ... 119
object operators ... 26
objects ... 13, 14, 15, 160, 273
and classes ... 13, 14, 15
data access ... 160
Form ... 273
octal ... 3
ODOMETER ... 640
SET ... 640
OEM() ... 854
OF ... 121
OLE ... 623, 863
REPLACE ... 623
OLE and binary data ... 5
OLE class ... 309
OleAutoClient class ... 866
oleType ... 415
ON ERROR ... 854
ON ESCAPE ... 96
ON KEY ... 97
ON NETERROR ... 855
ON PAGE ... 799
onAbandon ... 244
onAdvise ... 873
onAppend ... 245, 415
onCellPaint ... 416
onChange ... 245, 416
onChangeCancel ... 417
onChangeCommitted ... 417
onChar ... 418
onCheckBoxClick ... 419
onClick ... 419
onClose ... 246, 419
onDelete ... 246
onDesignOpen ... 420
onDragBegin ... 420
onDragEnter ... 421
onDragLeave ... 421
onDragOver ... 422
onDrop ... 423
onEdit ... 247
onEditLabel ... 424
onEditPaint ... 424
onExecute ... 874
onExpand ... 425
onFormSize ... 425
onGotFocus ... 425
onGotValue ... 247
onHelp ... 426
onInitiate ... 98
onInitMenu ... 99
onKey ... 426
onKeyDown ... 427
onKeyUp ... 428
onLastPage ... 429
onLeftDblClick ... 429
onLeftMouseDown ... 430
onLeftMouseUp ... 430
onLostFocus ... 430
onMiddleDblClick ... 431
onMiddleMouseDown ... 431
onMiddleMouseUp ... 431
onMouseMove ... 431
onMouseOut ... 432
onMouseOver ... 433
onMove ... 434
onNavigate ... 247, 434
onNewValue ... 874
onOpen ... 248, 435
onPage ... 505
onPaint ... 435
onPeek ... 875
onPoke ... 875
onProgress ... 248
onRender ... 505
onRightDblClick ... 435
onRightMouseDown ... 436
onRightMouseUp ... 436
onSave ... 249
onSelChange ... 436

onSelection 436
onSize 437
onTimer 730
onUnadvise 876
onUpdate 99
OPEN DATABASE 612
open() 249, 438, 541
operator 28, 29
  alias 28
  call indirection grouping 28
  macro 29
operators 20, 22, 23, 24, 25, 26, 659
  arithmetic 22, 23
  assignment 22
  comparison 25
  decrement 24
  increment 24
  local SQL 659
  logical 24
  numeric 24
  object 26
  string 22, 23
operators and symbols 5, 20
OR 24
ORDER 641
  SET 641
ORDER BY clause 670
ORDER() 612
orientation 507
OS() 542
OTHERWISE 139
output 506
outputFilename 506
P
PACK 613
packTable() 250
padvance 808
PAGE 798, 799
  EJECT 798
  ON 799
pageCount() 438
pageno 438, 809
PageTemplate class 485
PaintBox class 311
Paper 508
Paper size 508
paperSize 507
paperSource 507
Parameter class 172
parameter substitutions 662
PARAMETERS 139
params 250, 439
parent 141
parse() 730
password 188, 234, 235
  addPassword( ) method 188
  login( ) method 234
  loginString property 235
  passing a user id and password automatically 235
paste() 439
path 543, 552
  SET 552
pattern match 663
patternStyle 440
PAYMENT() 757
pbpage 809
PCOL 804
  SET 804
PCOL() 800
pcolno 810
pcopies 810
PCOUNT() 141
pdriver 811
PdxField class 173
pecode 811
peek() 876
peject 812
pen 440
penStyle 441
penWidth 441
pepage 812
persistent 442
pform 813
phoneticLink 442
PI() 758
picture 251, 442
PLAY SOUND 877
plength 813
plineno 814
ploffset 814
plus operator 22
Plus.exe.manifest 466
PlusRun.exe.manifest 466
point 762
  SET 762
pointers 12
  function 12
poke() 877
Popup 43, 68
  class 68
  CREATE 43
popupEnable 443
popupMenu 444
porientation 815
position 543
ppitch 815
pquality 816
Precedence 21
PRECISION 251, 762
  SET 762
prefixEnable 444

prepare() 251
Preprocessor 17, 33, 882, 891
  directive symbol 33
  directives 17
  identifiers 891
  overview 882
preRender 506
prevSibling 444
print() 445
printable 445
printer 507, 508, 797, 805
  CLOSE 797
  SET 805
printer object 495, 507
printerName 507
printerSource 507
PRINTJOB 800
PRINTSTATUS 801
PRINTSTATUS() 801
PRIVATE 141
problemTableName 252
PROCEDURE 124, 143, 150
  CLOSE 124
  SET 150
procedureName 252
procRefCount 143
PROGRAM 123
  CLEAR 123
program example 17
Program execution 16
program files 15
PROGRAM() 855
programming data types 5
programs 15
Progress class 312
PROJECT 44, 52
  CREATE 44
  MODIFY 52
proper noun format 780, 790
  convert string 780, 790
PROPER() 780
properties 273
  common 273
PROTECT 121, 856
  keyword of CLASS 121
PROW 805
  SET 805
PROW() 801
pscode 817
pspacing 817
PUBLIC 144
PushButton class 313
PUTFILE() 543
puts() 545
PV() 758

Q

queries 663, 664
  live 663
  updateable 663, 664
QUERY 44
  CREATE 44
Query class 174
QUIT 144
quotation mark symbols 30

R

radian from degrees 751
radians 760
  from degrees 760
RadioButton class 315
random number between 0 and 1 759
RANDOM() 759
rangeMax 445
rangeMin 446
rangeRequired 446
RAT() 781
ratio of circumference to diameter 758
RD 549
read() 547
readln() 547
readModal() 447
readOnly 253
RECALL 614
RECCOUNT() 614
RECNO() 615
reconnect() 877
records 667, 668, 669, 670, 671
  copying (local SQL) 671
  deleting (local SQL) 667
  inserting (local SQL) 668
  retrieving (local SQL) 668, 669, 670, 671
RECSIZE() 615
Rectangle class 316
REDEFINE 145
reExecute() 447
ref 253, 448
REFCOUNT() 145
REFRESH 616, 641
  SET 641
refresh() 253, 448
refreshAlways 448
refreshControls() 254
refreshRow() 254
REINDEX 616
reindex() 255
RELATION 642
  SET 642
RELATION() 616
RELEASE 146, 347
RELEASE AUTOMEM 617
RELEASE DLL 878
RELEASE OBJECT 147

release() ... 449
releaseAllChildren() ... 449
RELEASECALLBACK ... 879
remarks ... 16
remarks symbol ... 31
removeAll() ... 702
removeKey() ... 703
RENAME ... 547
RENAME TABLE ... 618
rename() ... 548
renameTable() ... 255
render() ... 510
rendering reports ... 482
renderOffset ... 510
REPLACE ... 618
REPLACE AUTOMEM ... 620
REPLACE BINARY ... 620
replace characters ... 787, 788
REPLACE FROM ARRAY ... 621
REPLACE MEMO ... 622
REPLACE OLE ... 623
replaceFromFile() ... 256
REPLICATE() function ... 782
replicate() method ... 782
REPORT ... 45
  CREATE ... 45
Report class ... 486
reportGroup ... 511
reportPage ... 511
reports ... 480, 482
  example ... 480
  rendering ... 482
  simple ... 480
reportViewer ... 511
ReportViewer class ... 317
REPROCESS ... 643
  SET ... 643
requery() ... 256
requestLive ... 257
required ... 257
reserved words ... 660
  local SQL ... 660
resize() ... 703
resolution ... 507
RESOURCE() ... 878
RESTORE ... 147
RESTORE IMAGE ... 879
restrictions ... 663, 664
  local SQL ... 663, 664
RESUME ... 857
retrieving data ... 668, 669, 670, 671
  local SQL ... 668, 669, 670, 671
retrieving records ... 668, 669, 670, 671
  local SQL ... 668, 669, 670, 671
RETRY ... 857
RETURN ... 148
right ... 450
RIGHT() function ... 783
right() method ... 783
rightTrim() ... 783
RLOCK() ... 624
rmargin ... 817
RMDIR ... 549
roamingUsersPath ... 99
ROLLBACK() ... 258, 625
rotate ... 512
round number to nearest integer ... 760
ROUND() ... 760
rowCount() ... 258
rowNo() ... 259
rowSelect ... 450
Rowset class ... 175
rowset navigation ... 241, 243
rowset property ... 259, 450
RTOD() ... 760
RTRIM() ... 784
rules ... 1
  language ... 1
RUN ... 549
RUN() ... 550

S

SAFETY ... 644
  SET ... 644
SAVE ... 148
SAVE TO clause ... 671
save() ... 260
saveRecord() ... 450
scale ... 260
scaleFontBold ... 451
scaleFontName ... 451
scaleFontSize ... 452
SCAN ... 625
scan() ... 707
scroll() ... 452
scrollBar ... 89, 90, 453
ScrollBar class ... 319
scrollHOffset ... 453
scrolling client components ... 452
scrollVOffset ... 454
seconds ... 727, 736
  set Date object ... 736
SECONDS() ... 731
security ... 846
SEEK ... 627
seek() ... 550, 628
SELECT ... 629, 668
SELECT() ... 454, 629
selectAll ... 454
selected ... 455
selected() ... 455
selectedImage ... 456
semicolon (symbol) ... 32

SEPARATOR 101, 763
SET 763
separator symbol 32
server (DDE) 880
serverName 456
SESSION 180, 261, 583
CREATE 583
Session class 180
session property 261
SET 54
SET ALTERNATE 802
SET AUTONULLFIELDS 54
SET AUTOSAVE 630
SET BELL 55
SET BLOCKSIZE 55
SET CENTURY 731
SET CONFIRM 101
SET CONSOLE 803
SET COVERAGE 56
SET CUAENTER 101
SET CURRENCY 761
SET DATABASE 630
SET DATE 732
set Date object 737
SET DATE TO 733
SET DBTYPE 631
SET DECIMALS 761
SET DELETED 631
SET DESIGN 57
SET DEVELOPMENT 57
SET DIRECTORY 551
SET ECHO 58
SET EDITOR 59
SET ENCRYPTION 858
SET EPOCH 733
SET ERROR 858
SET ESCAPE 102
SET EXACT 632
SET EXCLUSIVE 632
SET FIELDS 633
SET FILTER 635
SET FULLPATH 551
SET FUNCTION 103
SET HEADINGS 636
SET HELP 59, 880
SET HOURS 734
SET IBLOCK 60
SET INDEX 636
SET KEY TO 637
SET LDCHECK 859
SET LDCONVERT 860
SET LIBRARY 149
SET LOCK 638
SET MARGIN 803
SET MARK 734
SET MBLOCK 61
SET MEMOWIDTH 639
SET MESSAGE 104
SET NEAR 640
SET ODOMETER 640
SET ORDER 641
SET PATH 552
SET PCOL 804
SET POINT 762
SET PRECISION 762
SET PRINTER 805
SET PROCEDURE 150
SET PROW 805
SET REFRESH 641
SET RELATION 642
SET REPROCESS 643
SET SAFETY 644
SET SEPARATOR 763
SET SKIP 645
SET SPACE 806
SET STEP 61
SET TALK 62
SET TIME 734
SET TYPEAHEAD 104
SET UNIQUE 645
SET VIEW 646
SET() 152
setAsFirstVisible() 457
setByte() 784
setDate() 735
setFocus() 457
setHours() 735
setMinutes() 736
setMonth() 736
setRange() 261
setSeconds() 736
setTic() 457
setTicFrequency() 458
setTime() 737
SETTO() 152
setYear() 737
Shape class 320
shapeStyle 458
share 262
SHELL() 105
shortCut 106
shortName() 552
showFormatBar() 459
showMemoEditor() 459
showSelAlways 459
showSpeedTip 460
showTaskBarButton 460
SIGN() 764
simple data types 2
SIN() 764
sine value of an angle 764
size 708

size() ... 553
sizeable ... 461
SKIP ... 645, 646
  SET ... 645
slash (symbol) ... 31
SLEEP ... 106
Slider class ... 321
smallTitle ... 461
SORT ... 647
sort() ... 708
sortChildren() ... 461
sorted ... 462
SOUND ... 877
  PLAY ... 877
SOUNDEX() ... 785
source ... 263
Source Aliasing ... 519
sourceAliases ... 107
SPACE ... 806
  SET ... 806
SPACE() function ... 786
space() method ... 786
special characters ... 30, 31, 32, 33
speedBar property ... 108, 462
  _app object ... 108
  PushButton object ... 462
speedTip ... 463
SpinBox class ... 322
spinOnly ... 463
SQL ... 263, 658
SQLERROR() ... 860
SQLEXEC() ... 860
SqlField class ... 181
SQLMESSAGE() ... 861
SQRT() ... 764
square root ... 764
startPage ... 495, 512
startSelection ... 463
state ... 264
statement codeblock symbol ... 32
statement separator symbol ... 32
statements ... 10, 11, 664
  local SQL ... 664
STATIC ... 153
STATUS ... 48
  DISPLAY ... 48
statusMessage ... 464
StdOut ... 506
  report output to ... 506
step ... 464
STORE ... 154
STORE AUTOMEM ... 648
StoredProc class ... 182
STR() ... 786
streamChildren() ... 464
streamFrame ... 512
StreamFrame class ... 487
streamSource ... 513
StreamSource class ... 489
string ... 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792
  centering ... 769
  convert number to ... 786
  convert to proper noun format ... 780, 790
  convert to uppercase ... 790, 792
  convert uppercase to lowercase ... 780, 790
  derive substring ... 789
  finding numbers in ... 792
  formatting ... 791
  if alphabetic and lowercase first character ... 774
  if alphabetic first character ... 773
  if uppercase first character ... 774, 775
  no leading space ... 777, 780
  no trailing spaces ... 791
  number of characters ... 777, 778
  position within another string ... 768, 772
  removing trailing spaces ... 783, 784
  replace specified characters ... 787, 788
  return characters from end ... 783
  return repeatedly ... 782
  return specified number of space characters ... 786
  returning ASCII values of a ... 770, 771
  returning numeric value of ... 768
  returning phonetic differences from a ... 771
  skeleton matching ... 779
  soundex ... 785
  specified number of characters from beginning ... 776, 777
  starting position of last string ... 775, 781
String class ... 766
string data ... 2
string functions ... 663
  local SQL ... 663
string operators ... 22, 23
string symbols ... 30
strings ... 772, 784
  and Unicode manipulation ... 772, 784
STRUCTURE ... 49, 53, 579
  COPY ... 579
  DISPLAY ... 49
  MODIFY ... 53
STRUCTURE EXTENDED ... 580, 585
  COPY ... 580
  CREATE ... 585
STUFF() function ... 787
stuff() method ... 788
style ... 465
styles ... 89
  fonts ... 89
subclassing ... 14

dynamic ... 14
SubForm class ... 324
subscript() ... 711
SUBSTR() ... 789
substring match ... 663
substring() ... 663, 789
SUM ... 649
SUM() ... 662
supported statements ... 664
local SQL ... 664
suppressIfBlank ... 513
suppressIfDuplicate ... 514
SUSPEND ... 862
switch ... 11
symbol ... 32
symbols ... 19, 20, 30, 31, 32, 33
and syntax ... 19
symbols and operators ... 5
syntax ... 17, 18, 19
capitalization guidelines ... 19
conventions ... 17
example ... 19
notation ... 18
sysMenu ... 465
systemTheme ... 466
T
TabBox class ... 327
TABLE ... 581, 588, 618, 665, 668
ALTER ... 665
COPY ... 581
DELETE ... 588
DROP ... 668
RENAME ... 618
table properties ... 183
TableDef class ... 183
tableDriver ... 264
tableExists() ... 265
tableLevel ... 265
tableName ... 266
tables ... 574, 659, 665, 666, 668
add or drop fields (local SQL) ... 665
CLOSE ... 574
creating (local SQL) ... 666
deleting (local SQL) ... 668
naming (local SQL) ... 659
tabs ... 818
tabStop ... 466
TAG ... 588
DELETE ... 588
TAG() ... 650
TAGCOUNT() ... 650
TAGNO() ... 651
TALK ... 62
SET ... 62
TAN() ... 765
tangent ... 765
TARGET() ... 651
tempTable ... 266
terminate() ... 881
terminateTimerInterval ... 108
text ... 466
Text class ... 328
text streaming ... 792
TextLabel class ... 330
textLeft ... 467
themeState ... 108
THROW ... 154
tics ... 468
ticsPos ... 468
time ... 712, 727, 734, 737
SET ... 734
set Date object ... 737
time in milliseconds ... 740
returning ... 740
TIME() ... 553, 738
timeout ... 881
Timer class ... 715
timezone ... 727
title ... 514
TO ARRAY ... 581
COPY ... 581
toggle ... 469
toGMTString() ... 738
toLocaleString() ... 739
toLowerCase() ... 790
ToolBar class ... 69
ToolButton class ... 72
toolTips ... 469
top ... 469
topic ... 882
topMost ... 470
toProperCase() ... 790
toString() ... 739
TOTAL ... 652
toUpperCase() ... 790
tracking ... 514
trackJustifyThreshold ... 514
trackRight ... 109
trackSelect ... 470
TRANSFORM() ... 791
transparent ... 470
TreeItem class ... 331
TreeView class ... 333
TRIM() ... 663, 791
trueTypeFonts ... 507
try ... 11, 154
TTIME() ... 739
TTOC() ... 740
turn of century and SET EPOCH ... 733
TYPE ... 266, 554
type conversion ... 8, 9
automatic ... 8

explicit ... 9
see also converting ... 8
type property ... 266
Field object ... 266
Parameter object ... 266
TYPE() ... 157
TYPEAHEAD ... 78, 104
CLEAR ... 78
SET ... 104
U
UAC ... 110
unadvise() ... 882
uncheckedBitmap ... 109
uncheckedImage ... 470
undo() ... 471
Unicode ... 772, 784
manipulating strings ... 772, 784
unidirectional ... 267
UNION clause ... 671
UNIQUE ... 267, 645
SET ... 645
UNIQUE() ... 653
UNLOCK ... 654
unlock() ... 267
unprepare() ... 268
UNTIL ... 129
DO ... 129
upBitmap ... 471
UPDATE ... 268, 654, 672
update() ... 269
updateable queries ... 663, 664
UpdateSet class ... 184
updateWhere ... 269
UPPER() ... 663, 792
uppercase ... 774, 775, 780, 790, 792
convert string ... 790, 792
convert to lowercase ... 780, 790
first character in string ... 774, 775
USE ... 655
usePassThrough property ... 269
user ... 271
User Account Control ... 110
user() ... 271, 862
useTablePopup ... 472
useUACPaths ... 110
UTC() ... 740
V
VAL() ... 792
valid ... 472
valid names ... 6
VALIDDRIVE() ... 555
validErrorMsg ... 472
validRequired ... 473
value ... 271, 272, 473
variableHeight ... 515
variables ... 7, 8, 123
assigning ... 8
clearing ... 123
defined ... 7
using in expressions ... 8
version ... 272
VERSION() ... 862
vertical ... 474
vertical scroll bars ... 90
verticalJustifyLimit ... 515
VIEW ... 474, 646
SET ... 646
visible ... 475
visibleCount() ... 475
visual component properties ... 273
common ... 273
visualStyle ... 476
vScrollBar ... 476
W
WAIT ... 115
Web applications ... 486, 506
report output ... 506
reports ... 486
streaming reports to StdOut ... 506
web property ... 116
when ... 477
Where clause ... 670
while ... 11, 129
DO ... 129
width ... 478
windowMenu ... 116
Windows XP styles ... 466
windowState ... 478
with ... 11, 158
WORKAREA() ... 657
wrap ... 479, 819
write() ... 555
writeln() ... 556
X
Xbase ... 557, 558
aliases ... 558
command scope ... 558
filenames ... 557
XBase command elements ... 557, 558
common ... 557, 558
Y
year ... 728
year 2000 ... 733
and SET EPOCH ... 733
YEAR() ... 741
Z
ZAP ... 658
z-order ... 425, 477
Events firing order when a form is opened 425, 477