

# FlagShip



**Object Oriented  
Database  
Development System**

**Cross-Compatible to Unix,  
Linux and MS-Windows**

 **MULTISOFT**

**Release 7.1**

**Section**

**FS2**

## The whole FlagShip 7 manual consist of following sections:

<b>Section</b>	<b>Content</b>	<b>Pages</b>
<b>GEN</b>	General information: License agreement & warranty, installation and de-installation, registration and support	18
<b>LNG</b>	FlagShip language: Specification, database, files, language elements, multiuser, multitasking, FlagShip extensions and differences	176
<b>FSC</b>	Compiler & Tools: Compiling, linking, libraries, make, run-time requirements, debugging, tools and utilities	90
<b>CMD</b>	Commands and statements: Alphabetical reference of FlagShip commands, declarators and statements	486
<b>FUN</b>	Standard functions: Alphabetical reference of FlagShip functions	640
<b>OBJ</b>	Objects and classes: Standard classes for Get, Tbrowse, Error, Application, GUI, as well as other standard classes	368
<b>RDD</b>	Replaceable Database Drivers	38
<b>EXT</b>	C-API: FlagShip connection to the C language, Extend C System, Inline C programs, Open C API, Modifying the intermediate C code	160
<b>FS2</b>	Alphabetical reference of FS2 Toolbox functions	376
<b>QRF</b>	Quick reference: Overview of commands, functions and environment	40
<b>PRE</b>	Preprocessor, includes, directives	30
<b>SYS</b>	System info, porting: System differences to DOS, porting hints, data transfer, terminals and mapping, distributable files	42
<b>REL</b>	Release notes: Operating system dependent information, predefined terminals	8
<b>APP</b>	Appendix: Inkey values, control keys, ASCII-ISO table, error codes, dBase and FoxPro notes, forms	34
<b>IDX</b>	Index of all sections	42
<b>fsman</b>	The on-line manual contains all above sections, search function, and additionally last changes and extensions	variable



*multisoft Datentechnik, Munich, Germany*

Copyright (c) 1992..2009

All rights reserved



***Object Oriented Database Development System,  
Cross-Compatible to UNIX, Linux and MS-Windows***

## **Section FS2**

Manual release: 7.1

For the current program release see label on distribution disk and  
your Activation Card, or check on-line by issuing *FlagShip -version*

# Copyright

Copyright © 1992..2009 by multisoft Datentechnik, D-81545 Munich, Germany. All rights reserved worldwide. Manual authors: Jan V. Balek, Ibrahim Tannir, Sven Koester

No part of this publication may be copied or distributed, transmitted, transcribed, stored in a retrieval system, or translated into any human or computer language, in any form or by any means, electronic, mechanical, magnetic, manual, or otherwise; or disclosed to third parties without the express written permission of multisoft Datentechnik. Please see also "License Agreement", section GEN.2

Made in Germany. Printed in Germany.

# Trademarks

**FlagShip™** is trademark of multisoft Datentechnik. Other trademarks: dBASE is trademark of Borland/Ashton-Tate, Clipper of CA/Nantucket, FoxBase of Microsoft/Fox, UNIX of AT&T/USL/SCO, AIX of IBM, MS-DOS and MS-Windows of Microsoft. Other products named herein may be trademarks of their respective manufacturers.

# Headquarter Address

## **Headquarter:**

multisoft Datentechnik  
Harthäuser Str. 85  
81545 München  
Germany

**E-mail:** support@flagship.de  
support@multisoft.de  
sales@multisoft.de

**Telephone:** (+49-89) 6490040  
**Fax:** (+49-89) 6412974

**Web/Ftp:** http://www.fship.com  
ftp://mult-soft.de/pub

Call or e-mail multisoft for your local dealer or distributor

# FS2 Toolbox (Add-On Library)

---

<b>1FS2-Toolbox: Introduction</b> .....	<b>7</b>
<b>String Manipulation</b> .....	<b>8</b>
ADDASCII ().....	11
AFTERATNUM ().....	12
ASCIISUM ().....	14
ASCPOS ().....	15
ATADJUST ().....	16
ATNUM ().....	18
ATREPL ().....	20
ATTOKEN ().....	22
BEFORATNUM () .....	24
BLANK ().....	26
CENTER ().....	27
CHARADD ().....	29
CHARAND ().....	30
CHAREVEN ().....	31
CHARLIST ().....	32
CHARMIRR () .....	33
CHARMIX ().....	34
CHARNOLIST () .....	35
CHARNOT ().....	36
CHARODD () .....	37
CHARONE ().....	38
CHARONLY ().....	39
CHAROR ().....	40
CHARPACK ().....	42
CHARRELA () .....	44
CHARRELREP ().....	45
CHARREM () .....	46
CHARREPL ().....	47
CHARSORT ().....	49
CHARSPREAD ().....	51
CHARSWAP ().....	52
CHARUNPACK () .....	53
CHARXOR ().....	54
CHECKSUM ().....	56
COMPLEMENT () .....	58
COUNTLEFT ().....	59
COUNTRIGHT ().....	60
CRC16 ().....	61
CRYPT () .....	63
CSETATMUPA ().....	65
CSETREF ().....	66
EXPAND ().....	68
HEXTOSTR ().....	69

---

JUSTLEFT ().....	70
JUSTRIGHT ().....	71
LIKE ().....	72
LTOC ().....	73
MAXLINE ().....	74
NUMAT ().....	76
NUMLINE ().....	77
NUMTOKEN ().....	78
PADLEFT ().....	79
PADRIGHT ().....	80
POSALPHA ().....	81
POSCHAR ().....	82
POSDEL ().....	83
POSDIFF ().....	84
POSEQUAL ().....	85
POSINS ().....	86
POSLOWER ().....	87
POSRANGE ().....	88
POSREPL ().....	90
POSUPPER ().....	91
RANGEREM ().....	92
RANGEREPL ().....	94
REBALL ().....	96
REMLLEFT ().....	97
REMRIGHT ().....	98
REPLALL ().....	99
REPLLEFT ().....	101
REPLRIGHT ().....	102
RESTTOKEN ().....	103
SAVETOKEN ().....	104
SETATLIKE ().....	105
STRDIFF ().....	107
STRSWAP ().....	109
STRTOHEX().....	111
TABEXPAND ().....	112
TABPACK ().....	114
TOKEN ().....	116
TOKENARRAY ().....	118
TOKENAT ().....	120
TOKENEND ().....	121
TOKENINIT ().....	122
TOKENLOWER ().....	124
TOKENNEXT ().....	125
TOKENSEP ().....	127
TOKENSEPDEF ().....	128
TOKENUPPER ().....	129
VALPOS ().....	130
WORDONE ().....	131
WORDONLY ().....	132

WORDREPL ().....	133
WORDSWAP ().....	135
WORDTOCHAR ().....	136
<b>Numeric, Mathematical Functions .....</b>	<b>137</b>
<b>Mathematical Functions .....</b>	<b>139</b>
ACOS () .....	140
ASIN () .....	141
ATAN () .....	142
ATN2 () .....	143
CEILING () .....	144
COS () .....	145
COT () .....	146
DTOR () .....	147
FACT () .....	148
FLOOR () .....	149
FV () .....	150
GETPREC () .....	151
LOG10 () .....	152
PAYMENT () .....	153
PERIODS () .....	154
PI () .....	155
PV () .....	156
RATE () .....	157
RTOD () .....	158
SETPREC () .....	159
SIGN () .....	160
SIN () .....	161
TAN () .....	162
<b>Number and Bit Manipulation .....</b>	<b>163</b>
BITTOC () .....	164
CELSIUS () .....	165
CLEARBIT () .....	166
CTOBIT () .....	167
CTOF () .....	168
CTON () .....	169
EXPONENT () .....	171
FAHRENHEIT () .....	172
FTOC () .....	173
INFINITY () .....	174
INTNEG () .....	175
INTPOS () .....	176
ISBIT () .....	177
LTON () .....	178
MANTISSA () .....	179
NTOC () .....	180
NUMAND () .....	181
NUMCOUNT () .....	182

NUMHIGH () .....	183
NUMLOW () .....	184
NUMMIRR () .....	185
NUMNOT () .....	186
NUMOR () .....	187
NUMROL () .....	188
NUMXOR () .....	190
RAND () .....	191
RANDOM () .....	193
SETBIT () .....	195
<b>Date, Time Functions &amp; Triggers .....</b>	<b>196</b>
ADDMONTH () .....	197
BOM () .....	198
BOQ () .....	199
BOY () .....	200
CDOW2 () .....	201
CMONTH2 () .....	202
CTODOW () .....	203
CTOMONTH () .....	204
DMY () .....	205
DOY () .....	207
EOM () .....	208
EOQ () .....	209
EOY () .....	210
ISLEAP () .....	211
KEYSEC () .....	212
KEYTIME () .....	214
LASTDAYOM () .....	217
MDY () .....	218
NTOCROW () .....	220
NTOCMONTH () .....	221
QUARTER () .....	222
SECTOTIME () .....	223
SETDATE () .....	225
SETTIME () .....	226
SHOWTIME () .....	227
TIMETOSEC () .....	229
TIMEVALID () .....	231
TRIGGERUDF () .....	233
WAITPERIOD () .....	235
WEEK () .....	236
WEEKSETISO () .....	238
WOM () .....	239
<b>Windowing Functions .....</b>	<b>240</b>
Introduction .....	240
WACLOSE () .....	243
WBOARD () .....	244
WBOX () .....	245



WCAPTION () .....	247
WCENTER () .....	248
WCLOSE () .....	249
WCOL () .....	250
WFCOL () .....	252
WFLASTCOL () .....	253
WFLASTROW () .....	254
WFORMAT () .....	255
WFROW () .....	257
WLASTCOL () .....	258
WLASTROW () .....	259
WMAXCOL () .....	260
WMAXROW () .....	262
WMIDDLE () .....	264
WMODE () .....	265
WMOVE () .....	266
WMOVEDOWN () .....	267
WMOVELEFT () .....	268
WMOVERIGHT () .....	269
WMOVEUP () .....	270
WMOVEUSER () KSETSCROLL () .....	271
WNUM () .....	273
WOPEN () .....	274
WROW () .....	277
WSELECT () .....	278
WSETMOVE () .....	279
WSETSHADOW () .....	280
WSTEP () .....	281
<b>Video Functions and Extended Drivers .....</b>	<b>282</b>
Introduction .....	282
CLEAR_EOL () .....	285
CLEARWIN () .....	287
CLEOL () .....	289
CLWIN () .....	290
COLORTON () .....	291
DSETTYPE () .....	292
ENHANCED () .....	293
GETCLEARA () GETCLEARATTR () .....	294
GETCLEARB () GETCLEARBYTE () .....	295
FILESCREEN () .....	296
KEYSEND () .....	297
MAXCOL () MAXROW () .....	298
NUMCOL () .....	299
NTOCOLOR () .....	300
SCREENFILE () .....	301
SCREENSIZE () .....	303
SETCLEARA () SETCLEARATTR () .....	304
SETCLEARB () SETCLEARBYTE () .....	305

STANDARD ().....	306
TRAPANYKEY () .....	307
UNSELECTED () .....	309
<b>Serial Communication .....</b>	<b>310</b>
<b>Disk, File and Database Functions .....</b>	<b>312</b>
DELETEFILE () .....	314
DIRCHANGE () .....	316
DIRMAKE () .....	318
DIRNAME () .....	320
DIRREMOVE () .....	322
DISKFREE () .....	323
DISKTOTAL () .....	324
FILEAPPEND () .....	325
FILEATTR () .....	326
FILECHECK () .....	329
FILECOPY () .....	330
FILEDATE () .....	331
FILEDELETE () .....	332
FILEGROUP () .....	334
FILEMOVE () .....	335
FILEOWNER () .....	336
FILERIGHTS () .....	337
FILESEEK () .....	339
FILESIZE () .....	340
FILESTR () .....	341
FILESYMLNK () .....	343
FILETIME () .....	344
FILEVALID () .....	345
RENAMEFILE () .....	346
RESTFSEEK () .....	347
SAVEFSEEK () .....	348
SETFATTR () .....	351
SETFCREATE () .....	353
SETFDATI () .....	354
STRFILE () .....	356
TEMPFILE () .....	360
TRUENAME () .....	362
<b>Settings, System Functions.....</b>	<b>363</b>
<b>Index.....</b>	<b>367</b>

# FS2-Toolbox: Introduction

---

The FS2-Toolbox is an optional add-on library for FlagShip. The behavior and function syntax is compatible to Nantucket NT2 and CA-Tools3. Many of the FS2 Toolbox functions have additional functionality (compared to CA3), provided by optional parameters. These extensions are documented in the Compatibility section of each function reference.

To be able to use the FS2 Toolbox functions listed here, you need to have the FS2 Toolbox installed. Otherwise linker error (undefined symbol ...) occurs during the compiling/link stage.

There is nothing special required to use these functions. Once the FS2 Toolbox is installed, you will invoke its functions by the same way as the standard FlagShip functions. The required settings for the FS2 library is added automatically during the installation in the <FlagShip\_dir>/etc/ FS6config file, used by FlagShip for the compiling and linking stage (see section FSC).

# String Manipulation

---

## Introduction

The FS2 Toolbox string functions extends FlagShip with additional string handling, searching, tokenizing, crypting and further manipulation. While several functions are really complex and requires good IT knowledge, some other functions can also be emulated by the standard FlagShip functions like Substr(), At(), Rat(), Len(), Strpeek(), Strtran() and so on, but using these FS2 functions does not require additional programming effort, cost and overhead.

## Index of FS2 String Functions

ADDASCII()	Adds a value to the last or specified character in a string
AFTERATNUM()	Remainder of a string after nth appearance of sequence
ASCIIISUM()	Finds sum of the ASCII values of all string characters
ASCPOS()	Determines ASCII value of a character at a specif. position
ATADJUST()	Adjusts beginning position of a sequence within a string
ATNUM()	Determines starting position of a sequence within a string
ATREPL()	Searches for a sequence within a string and replaces it
ATTOKEN()	Finds the position of a token within a string
BEFORATNUM()	String segment before the nth occurrence of a sequence
BLANK()	Creates a blank/empty value for each data type
CENTER()	Centers a string using pad characters or string width
CHARADD()	Adds the corresponding ASCII codes of two strings
CHARAND()	Links corresponding ASCII codes of paired strings with AND
CHAREVEN()	Returns characters in the even positions of a string
CHARLIST()	Lists each character in a string
CHARMIRR()	Mirrors characters within a string
CHARMIX()	Mixes two strings together
CHARNOLIST()	Lists the characters that do not appear in a string
CHARNOT()	Complements each character in a string
CHARODD()	Returns characters in the odd positions of a string
CHARONE()	Reduces adjoining duplicate characters to 1 character
CHARONLY()	Determines the common denominator between two strings
CHAROR()	Joins the corresponding ASCII code of two strings with OR
CHARPACK()	Compresses (packs) a string using RLL or LZH algorithm
CHARRELA()	Correlates the character positions in paired strings
CHARRELREP()	Replaces chars in a string depend. on their correlation
CHARREM()	Removes particular characters from a string
CHARREPL()	Replaces certain characters with others
CHARSORT()	Sorts sequences within a string
CHARSPREAD()	Expands a string at the tokens
CHARSWAP()	Exchanges all adjoining characters in a string
CHARUNPACK()	Decompresses (unpacks) a string

CHARXOR()	Joins ASCII codes of paired strings with exclusive XOR
CHECKSUM()	Calculates the checksum for a character string (algorithm)
COMPLEMENT()	Forms the complement value of any data type
COUNTLEFT()	Counts a particular character at the beginning of a string
COUNTRIGHT()	Counts a particular character at the end of a string
CRYPT()	Encrypts and decrypts a string
CSETATMUPA()	Determines setting of the multi-pass mode for AT*() funct
CSETREF()	Determines whether call by value or by reference
EXPAND()	Expands a string by inserting characters
HEXTOSTR()	Converts a sting containing hex values into byte sequence
JUSTLEFT()	Moves characters from the beginning to the end of a string
JUSTRIGHT()	Moves characters from the end of a string to the beginning
LIKE()	Compares character strings using wildcard characters
LTOC()	Converts a logical value into a character
MAXLINE()	Finds the longest line within a string
NUMAT()	Counts the number of occurrences of a sequence in a string
NUMLINE()	Determines the number of lines required for string output
NUMTOKEN()	Determines the number of tokens in a string
PADLEFT()	Pads a string on the left to a particular length
PADRIGHT()	Pads a string on the right to a particular length
POSALPHA()	Determines position of first alphabetic char in a string
POSCHAR()	Replaces individual char at particular position in string
POSDEL()	Deletes characters at a particular position in a string
POSDIFF()	Finds the first position from which two strings differ
POSEQUAL()	Finds the first position at which two strings are the same
POSINS()	Inserts characters at a particular position within a string
POSLOWER()	Finds the position of the first lower case alphabetic char
POSRANGE()	Determines position of first char in an ASCII code range
POSREPL()	Replaces one or more characters from a certain position
POSUPPER()	Finds the position of the first uppercase, alphabetic char
RANGEREM()	Deletes chars that are within a specified ASCII code range
RANGEREPL()	Replaces characters within a specified ASCII code range
REALL()	Removes characters from the beginning and end of a string
REMLEFT()	Removes particular chars from the beginning of a string
REMRIGHT()	Removes particular chars at the end of a string
REPLALL()	Exchanges characters at the beginning and end of a string
REPLLEFT()	Exchanges particular chars at the beginning of a string
REPLRIGHT()	Exchanges particular chars at the end of a string
RESTTOKEN()	Recreates an incremental tokenizer environment
SAVETOKEN()	Saves the incremental tokenizer environment to a variable
SETATLIKE()	Provides an additional search mode for all AT functions
STRDIFF()	Finds similarity between two strings (Levenshtein Distance)
STRSWAP()	Interchanges two strings
STRTOHEX()	Converts a byte sequence into hexadecimal form
TABEXPAND()	Converts tabs to spaces
TABPACK()	Converts spaces in tabs
TOKEN()	Selects the nth token from a string

TOKENARRAY()	Create an array of all tokens in a string
TOKENAT()	Determines the most recent TokenNext() pos in a string
TOKENARRAY()	Returns an array of tokens
TOKENEND()	Determines if more tokens are available in TokenNext()
TOKENINIT()	Initializes a string for TokenNext()
TOKENLOWER()	Converts initial alphabetic char of a token into lowercase
TOKENNEXT()	Provides an incremental tokenizer
TOKENSEP()	Provides separator before/after last Token()
TOKENSEPDEF()	Return the list of default separators
TOKENUPPER()	Converts the initial letter of a token into upper case
VALPOS()	Determines numerical value of char at particular position
WORDONE()	Reduces multiple appearances of double characters to one
WORDONLY()	Finds common denominator of 2 strings on double char basis
WORDREPL()	Replaces particular double characters with others
WORDSWAP()	Exchanges double chars lying beside each other in a string
WORDTOCHAR()	Exchanges double chars for individual ones

# ADDASCII ()

---

**Syntax:**

**retC = AddAscii ( expC1, expN2, [expN3] )**

**Purpose:**

Adds an value to the last (or specified) character in a string.

**Arguments:**

<expC1> is the character string that is processed. Embedded zero bytes in <expC1> are supported.

<expN2> is the value that is added to the ASCII value the last character in <expC1> or to the character at position <expN3>

**Options:**

<expN3> is an optional position of the modified character within the <expC1> string. If not given, or is 0, the last character of <expC1> is used. When <expN3> is greater than LEN(expC1), or is less than 0, the string remain unchanged.

**Returns:**

<retC> is the modified string. Some of the resulting characters may contain zero byte chr(0) or chr(26) and if so, it is suggested to store <retC> in .dbt memo field by using MemoEncode(retC), see also LNG.4.2, FUN.MemoEncode() and CMD.REPLACE

**Description:**

With AddAscii(), you may modify any character of the given string by adding an ASCII value to it. If the resulting character value is > 255, only a modulo 256 of the result is used.

To subtract a value from the character value, use expN3 = 256 - value For example to replace lower case to upper case (within standard ASCII set), use <expN3> = 224, which subtracts 32 from the character value.

**Example:**

```
? AddAscii("Hello 123", 1)      // "Hello 124"
? AddAscii("Hello 123", 2, 0)   // "Hello 125"
? AddAscii("Hello 123", 2, 3)   // "Henlo 123"

myStr := "Hello 123"
cRet  := AddAscii(@myStr, 1)
? myStr                                // "Hello 123" = unchanged
? cRet                                // "Hello 124"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:** Substr(), Chr(), Asc()

# AFTERATNUM ()

---

## Syntax:

```
retC = AfterAtNum ( expC1, expC2, [expN3],  
                  [expN4] )
```

## Purpose:

Returns the rest of a string after the n-th or the last appearance of a found sequence.

## Arguments:

<expC1> is the string to be searched for in <expC2>

<expC2> is a string to be searched thru.

## Options:

<expN3> is an optional n-th occurrence counter of the found string. If not given, the last occurrence in the search expression is used.

<expN4> is an optional number of ignored characters by the search. The default value 0 ignores none.

## Returns:

<retC> is the remainder of the input string <expC2> behind the found n-th (or last) occurrence of <expC1>.

## Description:

This function finds the n-th (<expN3>) occurrence of <expC1> within <expC2> and returns the remainder of the string from the first position behind the located sequence.

AfterAtNum() start the search after skipping the first <expN4> characters in <expC2>. If the multi-pass switch CsetAtMupa() is set off (.F., the default), the search continues after the last character of the sequence most recently located. However, if the CsetAtMupa() is on, the search always continues after the first character in the most recently located sequence.

If the <expN4> parameter is not specified, the function initiates the search with the first character of <expC2>. If the <expN4> parameter is specified, <expN4> characters are ignored from the start of the string and are excluded from the search.

Embedded zero bytes in <expC1> or <expC2> are not considered and results in truncated <retC>.

## Example:

```
CsetAtMupa(.T.)           // the default  
? AtNum ("aa", "aBaaCaaX") // 7  
? AfterAtNum("aa", "aBaaCaaX") // "X"  
? AtNum ("aa", "aBaaCaaX", 1) // 3  
? AfterAtNum("aa", "aBaaCaaX", 1) // "CaaX"  
CsetAtMupa(.F.)  
? AtNum ("aa", "aBaaCaaX") // 6
```



```

? AfterAtNum("aa", "aBaaCaaaX")      // "aX"

//
//           1   1   2
// ..3.....9.....5...9...3.
String := " AxxBBBBBxxCCCCxxxDxxEEExx"

CsetAtMupa(.T.)                        // on, the default
? AtNum("xx", String)                  // 23
? AtNum("xx", String, 2)               // 9
? AtNum("xx", String, 2, 3)            // 15
? AtNum("xx", String, 3, 4)            // 16
? AfterAtNum("xx", String)             // ""
? AfterAtNum("xx", String, 2)          // "CCCCxxxDxxEEExx"
? AfterAtNum("xx", String, 2, 3)       // "xDxxEEExx"
? AfterAtNum("xx", String, 3, 4)       // "DxxEEExx"

CsetAtMupa(.F.)
? AtNum("xx", String)                  // 23
? AtNum("xx", String, 2)               // 9
? AtNum("xx", String, 2, 3)            // 15
? AtNum("xx", String, 3, 4)            // 19
? AfterAtNum("xx", String)             // ""
? AfterAtNum("xx", String, 2)          // "CCCCxxxDxxEEExx"
? AfterAtNum("xx", String, 2, 3)       // "xDxxEEExx"
? AfterAtNum("xx", String, 3, 4)       // "EEExx"

```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:AtNum(),      FS2:BeforeAtNum(),      FS2:CsetAtMupa(),      FS2:Token(),  
 FS2:TokenArray(), At(), Rat()

# ASCIISUM ()

---

**Syntax:**

**retN = AsciiSum ( expC1 )**

**Purpose:**

Calculates the sum of the ASCII values of all the characters of a string.

**Arguments:**

<expC1> is the character string that is processed. Embedded zero bytes in <expC1> are supported.

**Returns:**

<retN> is a number that corresponds to the sum of the ASCII codes of all the characters in <expC1>

**Description:**

AsciiSum() allows you to form simple checksums for character strings. For example, this can be implemented during remote data transmission to identify transmission errors.

**Example:**

```
? AsciiSum("abc")           // Result: 294
? AsciiSum("cba")           // Result: 294
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:Checksum(), Asc(), Substr()

# ASCPOS ()

---

**Syntax:**

**retN = AscPos ( expC1, [expN2] )**

**Purpose:**

Determines the ASCII value of a character at a particular position within a string.

**Arguments:**

<expC1> is the character string that is processed. Embedded zero bytes in <expC1> are supported.

**Options:**

<expN2> is an optional position of the character within <expC1>. If not given, the default is the last character, same as specifying LEN(expC1).

**Returns:**

<retN> is the Ascii value of the character at position <expN2>. If <expN2> is larger than the length of the string, 0 is returned.

**Description:**

With AscPos() you can determine the character value at particular position within a string. It is similar to using the standard function StrPeek(expC1,expN2) or to Asc(Substr(expC1,expN2,1))

**Example:**

```
? AscPos("abcdef", 5)           // 101
? strpeek("abcdef",5)           // 101
? asc(substr("abcdef",5,1))      // 101

str := "abcdef"
? AscPos(@str)                   // 102
? strpeek(@str,6)                // 102
? asc(right(@str,1))             // 102

str := "abcdef" + chr(0) + "gh"
? AscPos(@str)                   // 102
? AscPos(@str,9)                 // 0
? strpeek(@str,9)                // 104
? asc(right(@str,1))             // 104
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:ValPos(), Asc(), StrPeek()

# ATADJUST ()

---

**Syntax:**

```
retC = AtAdjust ( expC1, expC2, [expN3],  
                  [expN4], [expN5], [expNC6] )
```

**Purpose:**

Adjusts the beginning position of a sequence within a string.

**Arguments:**

<expC1> is the string to be searched for in <expC2>

<expC2> is a string to be searched thru.

**Options:**

<expN3> is an optional numeric value specifying from which position within <expC2> the search expression is adjusted. If not given, the search starts at position 1.

<expN4> is an optional numeric value specifying the occurrence of <expC1> is taken into account. The default value is for the last occurrence.

<expN5> is an optional numeric value specifying the number of characters at the beginning of the search string that are removed. The default value is 0 = none.

<expNC6> is either a character or a numeric equivalence of a character, to carry out the adjustment. The default value is a space = CHR(32)

**Returns:**

<retC> is the modified string or "" if error occurs.

**Description:**

AtAdjust() first looks for the <expC1> within the <expC2> string. From this point, the rest of the <expC2> is moved (adjusted) by either inserting or removing blanks until the <expN3> is reached. In lieu of blanks, <expNC6> can be used as a fill character.

Additionally you can specify that the n-th occurrence of <expC1> be used and whether or not a specific number of characters at the beginning of the search string is eliminated.

If the multi-pass switch CsetAtMupa() is set off (.F., the default), the search continues after the last character of the sequence most recently located. However, if the CsetAtMupa() is on, the search always continues after the first character in the most recently located sequence.

Embedded zero bytes in <expC1> and <expC2> are not considered and hence terminates the string.

**Example:**

```
cLine := "abcd < xyz"
? AtAdjust("<", cLine, 20)           // "abcd < xyz"
? AtAdjust("<", cLine, 20, , ".")     // "abcd .....< xyz"

CsetAtMupa(.F.)                      // off, the default
? AtAdjust("AA", "123AAABBB", 7, 2)  // "123  AAABBB"
CsetAtMupa(.T.)
? AtAdjust("AA", "123AAABBB", 7, 2)  // "123A  AABBB"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:CsetAtMupa(), FS2:SetAlike(), Strtran()

# ATNUM ()

---

## Syntax:

**retN = AtNum ( expC1, expC2, [expN3], [expN4] )**

## Purpose:

Search for a substring and returns the found position of the last or n-th occurrence of.

## Arguments:

<expC1> is the string to be searched for in <expC2>

<expC2> is a string to be searched thru.

## Options:

<expN3> is an optional n-th occurrence counter of the found string. If not given, the last occurrence in the search expression is used.

<expN4> is an optional number of ignored characters by the search. The default value 0 ignores none.

## Returns:

<retN> is the position of <expC1> in <expC2> where the n-th (or last) occurrence was found. If not found, 0 is returned.

## Description:

AtNum() start the search after skipping the first <expN4> characters in <expC2>. If the multi-pass switch CsetAtMupa() is set off (.F., the default), the search continues after the last character of the sequence most recently located. However, if the CsetAtMupa() is on, the search always continues after the first character in the most recently located sequence.

If the <expN4> parameter is not specified, the function initiates the search with the first character of <expC2>. If the <expN4> parameter is specified, <expN4> characters are ignored from the start of the string and are excluded from the search.

Embedded zero bytes in <expC1> and <expC2> are not considered and hence terminates the string.

## Example:

```
CsetAtMupa(.T.)           // the default
? AtNum ("aa", "aBaaCaaX") // 7
? AfterAtNum("aa", "aBaaCaaX") // "x"
? AtNum ("aa", "aBaaCaaX", 1) // 3
? AfterAtNum("aa", "aBaaCaaX", 1) // "CaaX"
CsetAtMupa(.F.)
? AtNum ("aa", "aBaaCaaX") // 6
? AfterAtNum("aa", "aBaaCaaX") // "aX"

//                               1 1 2
// ..3.....9.....5...9...3.
String := " AxxBBBBBxxCCCCxxxDxxEExx"
```

```

CsetAtMupa(.T.)                                // on, the default
? AtNum("xx", String)                          // 23
? AtNum("xx", String, 2)                       // 9
? AtNum("xx", String, 2, 3)                    // 15
? AtNum("xx", String, 3, 4)                    // 16
? AfterAtNum("xx", String)                     // ""
? AfterAtNum("xx", String, 2)                  // "CCCCxxxDxxEEExx"
? AfterAtNum("xx", String, 2, 3)               // "xDxxEEExx"
? AfterAtNum("xx", String, 3, 4)               // "DxxEEExx"

CsetAtMupa(.F.)
? AtNum("xx", String)                          // 23
? AtNum("xx", String, 2)                       // 9
? AtNum("xx", String, 2, 3)                    // 15
? AtNum("xx", String, 3, 4)                    // 19
? AfterAtNum("xx", String)                     // ""
? AfterAtNum("xx", String, 2)                  // "CCCCxxxDxxEEExx"
? AfterAtNum("xx", String, 2, 3)               // "xDxxEEExx"
? AfterAtNum("xx", String, 3, 4)               // "EEExx"

? Rat("xx", String)                           // 23
? At ("xx", String)                           // 3

```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:AfterAtNum(), FS2:CsetAtMupa(), FS2:Token(), At(), Rat()

# ATREPL ()

---

## Syntax:

```
retC = AtRepl ( expC1, expC2, expC3, [expN4],  
               [expL5] )
```

## Purpose:

Searches for a sequence within a string and replaces it.

## Arguments:

<expC1> is the string to be searched for in <expC2>

<expC2> is a string to be searched thru.

<expC3> is the character or string that is exchanged in <expC2> for the found sequence of <expC1>.

## Options:

<expN4> is optional occurrence counter, specifying which or how many occurrences of <expC1> within <expC2> are replaced by <expC3>. The default value is the last occurrence.

<expL5> is an optional repetition flag. If .F. (the default), all occurrences of <expC1> within <expC2> are replaced by <expC3>. If .T., only the <expN4>-th or the last occurrence is replaced.

## Returns:

<retC> is the modified string.

## Description:

AtRepl() work similarly to the standard Strtran() function, but has additionally the n-th occurrence feature and consider the CsetAtMupa() setting.

If the multi-pass switch CsetAtMupa() is set off (.F., the default), the search continues after the last character of the sequence most recently located. If the CsetAtMupa() is on, the search always continues after the first character in the most recently located sequence.

Embedded zero bytes in <expC1>, <expC2> and <expC3> are not considered and hence terminates the string.

## Example:

```
CsetAtMupa(.F.) // off, the default  
? AtRepl("aa", "aaaa", "a") // "aa"  
? AtRepl("abc", "123abcc456", "ab") // "123abc456"  
  
? AtRepl("123", "123_123_123", "ab") // "ab_ab_ab"  
? AtRepl("123", "123_123_123", "ab", 2) // "ab_ab_123"  
? AtRepl("123", "123_123_123", "ab", 2, .T.) // "123_ab_123"  
  
CsetAtMupa(.T.)  
? AtRepl("aa", "aaaa", "a") // "a"  
? AtRepl("abc", "123abcc456", "ab") // "123ab456"
```



```
? AtRepl("123", "123_123_123", "ab")           // "ab_ab_ab"
? AtRepl("123", "123_123_123", "ab", 2)         // "ab_ab_123"
? AtRepl("123", "123_123_123", "ab", 2, .T.)    // "ab_123_123"

? Strtran("123_123_123", "123", "ab")           // "ab_ab_ab"
? Strtran("123_123_123", "123", "ab", 2)        // "123_ab_ab"
? Strtran("123_123_123", "123", "ab", 2, 1)     // "123_ab_123"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:AtNum(), FS2:CsetAtMupa(), Strtran()

# ATTOKEN ()

---

## Syntax:

```
retN = AtToken ( expC1, [expC2], [expN3] )
```

## Purpose:

Finds the position of a token within a string.

## Arguments:

<expC1> is the string that is processed. Embedded zero bytes are supported.

## Options:

<expC2> is a list of delimiters used to identify the tokens. If not specified, the default <expC2> delimiters are: " ,.;!/?\<>()#^%&+~" + chr(9) + chr(13) + chr(10) + chr(26) + chr(138) + chr(141) + chr(4) + chr(0). Embedded zero bytes are supported.

<expN3> is optional occurrence counter, specifying which token position in <expC1> is determined. The default value is the last occurrence.

## Returns:

<retN> is the position of <expN3>-th (or last) token within the string <expC1>, starting at 1. If there is no (more) token(s) available, 0 is returned.

## Description:

AtToken() work similarly to the standard At() or AtAnyChar() functions, but has additionally the n-th occurrence feature and pre-defined list of delimiters.

## Example:

```
? AtToken("Have a nice day.")           // 13
? AtToken("Have a nice day.", , 1)       // 1
? AtToken("Have a nice day.", , 2)       // 6
? AtToken("Have a nice day.", , 3)       // 8
? AtToken("Have a nice day.", , 6)       // 0
? AtToken("Have a nice day.", "dy")      // 16
? AtToken("Have a nice day.", "yd")      // 16
```

## Example 2:

Get an array of tokens, here single words of a sentence:

```
local str := "what a beautiful day today, isn't it?"
aTokens := TokenArray(@str)
aeval(aTokens, {|x| qout(x)}) // print it
```

## Example 3:

Same as Example 2, simulates TokenArray() by AtToken():

```
local str := "what a beautiful day today, isn't it?"
local delim := " ,.;?!"
local aTokens := {}
local iOccur, iPos1, iPos2

for iOccur := 1 to 1000
    iPos1 := AtToken(@str, @delim, iOccur)
    if iPos1 == 0
```

```

        exit
    endif
    iPos2 := AtAnyChar(@delim, substr(@str,iPos1))
    if iPos2 > 1
        aadd(aTokens, substr(@str, iPos1, iPos2 -1))
    else
        aadd(aTokens, substr(@str, iPos1))
    endif
next
aeval(aTokens, {|x| qout(x)}) // print it

```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:AtNum(), FS2:Token(), FS2:NumToken(), FS2:TokenArray(), At(), Rat(), AtAnyChar()

# BEFORATNUM ()

---

## Syntax:

```
retC = BeforAtNum ( expC1, expC2, [expN3],  
                  [expN4] )
```

## Purpose:

Returns the string segment before the n-th occurrence of a sequence

## Arguments:

<expC1> is the string to be searched for in <expC2>

<expC2> is a string to be searched thru.

## Options:

<expN3> is an optional n-th occurrence counter of the found string. If not given, the last occurrence in the search expression is used.

<expN4> is an optional number of ignored characters by the search. The default value 0 ignores none.

## Returns:

<retC> are all the characters of <expC2> in front of the sequence <expC1> determined, or "" if nothing is found.

## Description:

This function finds the n-th (<expN3>) occurrence of <expC1> within <expC2> and returns the leading string up to the found sequence.

BeforAtNum() start the search after skipping the first <expN4> characters in <expC2>. If the multi-pass switch CsetAtMupa() is set off (.F., the default), the search continues after the last character of the sequence most recently located. If the CsetAtMupa() is on, the search always continues after the first character in the most recently located sequence.

If the <expN4> parameter is not specified, the function initiates the search with the first character of <expC2>. If the <expN4> parameter is specified, <expN4> characters are ignored from the start of the string and are excluded from the search.

Embedded zero bytes in <expC1> or <expC2> are not considered and results in truncated <retC>.

## Example:

```
? BeforAtNum("ab", "abcabdabe")           // "abcabd"
? BeforAtNum("ab", "abcabdabe", 1)         // ""
? BeforAtNum("ab", "abcabdabe", 1, 3)      // "abc"

String := "AxxBBBBBxxCCCCxxxDxxEExx"
CsetAtMupa(.T.)
? BeforAtNum("xx", String, 3, 4)           // "AxxBBBBBxxCCCCx"
CsetAtMupa(.F.)
? BeforAtNum("xx", String, 3, 4)           // "AxxBBBBBxxCCCCxxxD"
```

```

CsetAtMupa(.T.)                                // the default
? AtNum      ("aa", "aBaaCaaaX")                // 7
? AfterAtNum("aa", "aBaaCaaaX")                // "X"
? BeforAtNum("aa", "aBaaCaaaX")                // "aBaaCa"

? AtNum      ("aa", "aBaaCaaaX", 1)            // 3
? AfterAtNum("aa", "aBaaCaaaX", 1)            // "CaaaX"
? BeforAtNum("aa", "aBaaCaaaX", 1)            // "aB"

CsetAtMupa(.F.)
? AtNum      ("aa", "aBaaCaaaX")                // 6
? AfterAtNum("aa", "aBaaCaaaX")                // "aX"
? BeforAtNum("aa", "aBaaCaaaX")                // "aBaaC"

? AtNum      ("aa", "aBaaCaaaX", 1)            // 3
? AfterAtNum("aa", "aBaaCaaaX", 1)            // "CaaaX"
? BeforAtNum("aa", "aBaaCaaaX", 1)            // "aB"

```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:AtNum(),      FS2:AfterAtNum(),      FS2:Token(),      FS2:TokenArray(),  
 FS2:CsetAtMupa(), At(), Rat()

# BLANK ()

---

## Syntax:

**ret = Blank ( exp1, [expL2] )**

## Purpose:

Creates a blank/empty value for each data type.

## Arguments:

<exp1> is any valid expression of type C/N/L/D/A/B/O/U for which an empty or blank value should be created.

## Options:

<expL2> designates whether spaces or an empty string should be created when <exp1> is of type "C". The default is .F. and creates null-string "".

## Returns:

<ret> is an empty value (or spaces) of the same type as <exp1> or .F. on error.

## Description:

The Blank() function acts similarly to the standard FS function SetVarEmpty(), but Blank() does not change the input variable <exp1>. It returns an empty value for:

valtype(exp1)	<ret> value
C = character	"" or Space(Len(exp1))
N = numeric	0.0
L = logical	.F.
D = date	empty date, same as Ctod(" / / ")
A = array	empty array, same as Array(0)
B = code block	empty code block, same as {   NIL }
O = object	empty object
U = NIL	NIL
other	.F.

## Example:

```
? Blank(Date())           // " / / "
? Blank()                 // .F.
? Blank(.T.)              // .F.
? Blank(99)               // 0
? Blank("abc")            // ""
? Blank("abc", .T.)       // " "
? Blank( {1,2} )          // empty array
? Blank( {|x| qout(x)} )  // empty code block
? Blank( GetNew() )       // empty object
```

**Classification:** add-on library, programming

**Compatibility:** Available in FS2-Toolbox, compatible to NT2/CA3 tools, which supports only types C/N/L/D.

**Related:** LOCAL..AS, Empty(), SetVarEmpty(), Space(), Valtype()

# CENTER ()

---

**Syntax:**

```
retC = Center ( expC1, [expN2], [expC3], [expLC4] )
```

**Purpose:**

Centers a string using pad characters.

**Arguments:**

<expC1> is the character string that is processed.

**Options:**

<expN2> is the length of the line within which the <expC1> sequence is centered. If not specified, MaxCol() +1 is used. When <expLC4> is character based, <expN2> is the width in columns, which is to be filled.

<expC3> is an optional character used as filler. If not specified, blank " " is used.

<expLC4> is either an optional logical value specifying whether only the beginning, or both sides of <expC1> are padded. The default value (.F.) only fills the beginning. You also may specify a string containing

"L"	= pad left in full size of <expN2>
"LH"	= pad left only (half padding), similar to .F. or NIL
"R"	= pad right in full size of <expN2>
"RH"	= pad right only (half padding)
"C" or "LR" or ""	= center, pad left and right, similar to .T.

The logical <expLC4> entry is compatible to CA3 Tools, and Center() is working on character basis. Specifying <expLC4> as string gives you more flexibility and Center() will also consider proportional fonts in GUI i/o mode (if such), calculating the real column and <expC1>, <expC3> width, instead using the character count.

**Returns:**

<retC> is the modified string. If the size of <expC1> exceeds the size of <expN2>, trimmed but un-truncated <expC1> is returned.

**Description:**

This function provides a simple way to center text in any line. Center() is able to pad only on the left or on the left and right using any character of your choice. Leading and trailing blanks or the <expC3> characters resp. (if any) are removed before the padding calculation starts.

Embedded zero bytes in <expC1> are not considered and results in truncated <retC>.

If a proportional font is used in GUI, the size of the <expC1> and the filler size is considered correspondingly, when <expLC4> is given as string. See the example below showing the difference.

**Example:**

```

? "with logical/none param4" color "R+" guicolor "r+"
? Center("xx", 10) + " | def" // " xx"
? Center("..xx...", 10, ".", .F.) + " | .F." // "...xx"
? Center("xx", 10, , .T.) + " | .T." // " xx "
? Center("xx", 10, , .T.) + " | .T." // " xx "
? Center("xx", 10, ".", .T.) + " | .T." // "...xx..."
? Center("x", 10, ".", .T.) + " | .T." // "....x...."

? "with param4 as character" color "R+" guicolor "r+"
? Center("xx", 10, , "LH") + " | LH" // " xx"
? Center("..xx...", 10, ".", "LH") + " | LH" // "...xx"
? Center("xx", 10, , "C") + " | C" // " xx "
? Center("xx", 10, , "RL") + " | RL/C" // " xx "
? Center("xx", 10, ".", "C") + " | C" // "...xx..."
? Center("x", 10, ".", "C") + " | C" // "....x...."

? Center("xx", 10, , "L" ) + " | L" // " xx"
? Center("xx", 10, , "R" ) + " | R" // "xx "
? Center(" xx", 10, , "LH") + " | LH" // " xx"
? Center(" xx", 10, , "RH") + " | RH" // "xx "

wait
cls
// SET FONT TO "Helvetica", 12
? "You will see difference here only with proportional char set"
for rr := 4 to 7
  for cc := 0 to 40
    @ rr,cc say "X"
  next
next
/* The first Center() fills upto 20 characters (logical expLC4),
* the second Center() fills upto the width of 20 columns.
* Both are equivalent in Terminal i/o but may differ in GUI
*/
@ 5, 10 SAY Center("FlagShip", 20, ".", .T.) ;
color "R+/B" guicolor "R+/W-"
@ 6, 10 SAY Center("FlagShip", 20, ".", "C") ;
color "GR+/B" guicolor "GR+/W-"

setpos(9,0)
wait

```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools does not support the <expLC4> argument as string and hence work on character basis only.

**Related:**

FS2:PadLeft(), FS2:PadRight(), Padl(), Padr()



# CHARADD ()

---

## Syntax:

**retC = CharAdd ( expC1, expC2 )**

## Purpose:

Adds the ASCII value(s) of string2 to all characters of string1.

## Arguments:

**<expC1>** is the character string that is processed. Embedded zero bytes in <expC1> are supported.

**<expC2>** is the character string whose value is added to <expC1> Embedded zero bytes in <expC2> are supported.

## Returns:

**<retC>** is the modified string. Some of the resulting characters may contain zero byte chr(0) or chr(26) and if so, it is suggested to store <retC> in .dbt memo field by using MemoEncode(retC), see also LNG.4.2, FUN.MemoEncode() and CMD.REPLACE

## Description:

You can use CharAdd() to produce simple character string coding, manipulation or simple crypting (although CharXor() is more sinful for crypting and Crypt() is available as well in FS2).

When both character strings are the same length, then the first byte of <expC1> can be linked with the first byte of <expC2>, and the second byte of <expC1> can be linked with the second byte of <expC2>, and so on. If <expC2> is shorter than <expC1>, then as soon as the last byte of <expC2> is reached, the process continues and starts again with the first byte of <expC2>. However, if <expC1> is shorter than <expC2>, the process terminates with the end of <expC1>.

Values greater than 256 can result when adding values which then results in modulo 256, or the resulting character value can be calculated the following formula:  $(ASC(char\_of\_expC1) + ASC(char\_of\_expC2)) \% 256$

## Example:

```
? CharAdd("01234abc", CHR(1))           // "12345bcd"
? CharAdd("12345bcd", CHR(255))         // "01234abc"
? CharAdd("11111111", "123")            // "bcdbcdbc"
? CharAdd("HELLO WORLD", CHR(32))        // "hello@world"
? CharAdd("hello@world", CHR(256-32))    // "HELLO WORLD"
```

**Classification:** add-on library, programming

**Compatibility:** Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:** FS2:CharAnd(), FS2:CharOr(), FS2:CharXor(), FS2:AddAscii(), Asc(), Chr(), Substr()

# CHARAND ()

---

## Syntax:

**retC = CharAnd ( expC1, expC2 )**

## Purpose:

Links the ASCII value(s) of string2 to all characters of string1 by using binary AND operation.

## Arguments:

<expC1> is the character string that is processed. Embedded zero bytes in <expC1> are supported.

<expC2> is the character string whose value is AND-ed to <expC1> Embedded zero bytes in <expC2> are supported.

## Returns:

<retC> is the modified string. Some of the resulting characters may contain zero byte chr(0) or chr(26) and if so, it is suggested to store <retC> in .dbt memo field by using MemoEncode(retC), see also LNG.4.2, FUN.MemoEncode() and CMD.REPLACE

## Description:

You can use CharAnd() to produce simple character string coding, manipulation, reset the high bit for all the characters in a string or simple crypting (although CharXor() is more sinful for crypting and Crypt() is available as well in FS2).

CharAnd() joins (links byte by byte using binary AND operation) each character in <expC1> with the corresponding character in <expC2>. When both character strings are the same length, then the first byte of <expC1> can be linked with the first byte of <expC2>, and the second byte of <expC1> can be linked with the second byte of <expC2>, and so on. If <expC2> is shorter than <expC1>, then as soon as the last byte of <expC2> is reached, the process continues and starts again with the first byte of <expC2>. However, if <expC1> is shorter than <expC2>, the process terminates with the end of <expC1>.

## Example:

```
? CharAnd("11111111", "123")           // "10110110"
? CharAnd("01234567", "1" )             // "01010101"
? CharAnd("01234xyz", "123")            // "00210012"
? CharAnd("123abc ", "123")              // "123!"# "
```

## Classification:

add-on library, programming

## Compatibility:

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

## Related:

FS2:CharAdd(), FS2:CharOr(), FS2:CharXor(), FS2:AddAscii(), Asc(), Chr(), Substr()

# CHAREVEN ()

---

**Syntax:**

`retC = CharEven ( expC1 )`

**Purpose:**

Returns characters in the even positions of a string.

**Arguments:**

<expC1> is the character string that is processed.

**Returns:**

<retC> is the modified string.

**Description:**

CharEven() returns a string containing all the even characters of <expC1>. It assembles all even characters within a string into a new string. The first position in a string is 1, and therefore is not even.

Embedded zero bytes in <expC1> are not considered and hence terminates the string.

**Example:**

```
? CharEven("1234a")           // "24"  
? CharEven(" H e l l o")       // "Hello"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:CharOdd(), FS2:CharMix()

# CHARLIST ()

---

**Syntax:**

**retC = CharList ( expC1 )**

**Purpose:**

Creates a list of characters used in a string.

**Arguments:**

<expC1> is the character string that is processed.

**Returns:**

<retC> is the resulting list. The size of <retC> is never longer than 255 characters. The resulting string may contain chr(26) and if so, it is suggested to store <retC> in .dbt memo field by using MemoEncode(retC), see LNG.4.2, FUN.MemoEncode() and CMD.REPLACE

**Description:**

CharList() determines the list of characters that appear in the <expC1>. Each character only appears in the list once. You can sort this result with CharSort() to get an ascending list of characters.

Embedded zero bytes in <expC1> are not considered and hence terminates the string.

**Example:**

```
? CharList("Hello goodbye")           // "HeLo gdby"
? CharSort(CharList("Hello goodbye"))  // " Hbdegloy"
str := "what a beautiful day today, isn't it?"
? CharList(@str)                       // "what beuifldyo,sn'?"
? CharSort(CharList(@str))             // " ',?wabdefhlnostuy"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:CharNolist(), FS2:CharOne(), FS2:CharSort()

# CHARMIRR ()

---

**Syntax:**

```
retC = CharMirr ( expC1, [expL2] )
```

**Purpose:**

Mirrors characters within a string.

**Arguments:**

<expC1> is the character string that is processed.

**Options:**

<expL2> is an optional logical flag, specifying whether to mirror everything (.F., the default) or not to mirror the blanks at the end of the string <expC1>

**Returns:**

<retC> is the mirrored string.

**Description:**

CharMirr() reverses a string. The function returns a palindrome of <expC1>. The optional <expL2> flag permits you to build index entries that end with a particular sequence.

Embedded zero bytes in <expC1> are not considered and results in truncated <retC>.

**Example:**

```
? CharMirr("abc321")           // "123cba"
? CharMirr("Hello world ")     // " dlrow olleH"
? CharMirr("Hello world ", .T.) // "dlrow olleH "

USE mydbf
INDEX on lower(CharMirr(text, .T.)) TO myidx
append blank
replace text with "mayflower " // index key = "rewolfyam "
append blank
replace text with "Magic power " // index key = "rewop cigam "
SEEK "rewo" // will find both records
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

Strpeek(), Substr()

# CHARMIX ()

---

**Syntax:**

```
retC = CharMix ( expC1, expC2 )
```

**Purpose:**

Mixes two strings together.

**Arguments:**

<expC1> is the character string that is mixed with <expC2>.

<expC2> is the character string that is mixed with <expC1>.

**Returns:**

<retC> is the mixed string. The length is twice of <expC1> assuming <expC1> contains at least one character.

**Description:**

CharMix() mix the characters from two strings together. The characters from <expC1> and <expC2> appear alternately in the result string. The length of <expC1> forms the basis for the maximum count of the resulting mix. A longer <expC2> string is cut down to the length of <expC1>. A shorter <expC2> is processed from beginning to end, wrapping to the beginning again until there are no more characters in <expC1>. This function can be used to recombine strings extracted by CharEven() or CharOdd().

Embedded zero bytes in <expC1> and <expC2> are not considered and hence terminates the string.

**Example:**

```
? CharMix("ABC", "123")      // "A1B2C3"
? CharMix("ABCDE", "12")     // "A1B2C1D2E1"
? CharMix("AB", "12345")     // "A1B2"
? CharMix("HELLO", " ")      // "H E L L O "
? CharMix("HELLO", "")       // "HELLO"
? CharMix("", "1234")        // ""
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:CharEven(), FS2:CharOdd(), FS2:Expand(), Substr()

# CHARNOLIST ()

---

**Syntax:**

**retC = CharNolist ( [expC1] )**

**Purpose:**

Lists the characters that do not appear in a string.

**Options:**

<expC1> is the character string that is processed. If not specified, "" is assumed.

**Returns:**

<retC> is the resulting list. The size of <retC> is never longer than 255 characters. The resulting string may contain chr(26) and if so, it is suggested to store <retC> in .dbt memo field by using MemoEncode(retC), see LNG.4.2, FUN.MemoEncode() and CMD.REPLACE

**Description:**

CharNolist() determines the list of characters that does not appear in the <expC1>. It is a counterpart of CharList() function.

Embedded zero bytes in <expC1> are not considered and hence terminates the string.

**Example:**

```
// Delete all characters except "XYZ":
cString := "ABXCDYEF"
? CharRem(CharNolist("XYZ"), cString)      // "XY"

// Generate a list of all 255 ASCII characters (1..255):
cList := CharNolist()
? len(cList),"=",cList                     // 255 = ..... !"#%&'()*+,-
./0123....

// Replace CHR(0) and CHR(26) in text by unused characters.
// This can be useful for memo field, where all zero bytes and
// CHR(26) are ignored and will terminate the data. This roughly
// simulates MemoEncode() and MemoDecode() standard functions which
// do this automatically and also remember the replacement.
cMemoText := "Abcd" + chr(26) + "efg" + chr(0) + "ijk"
cNoList    := CharNolist(@cMemoText)
cNoChar1   := substr(cNoList, 1, 1)        // == chr(1)
cNoChar2   := substr(cNoList, 2, 1)        // == chr(2)
cMemoText  := CharRepl(CHR(26), @cMemoText, cNoChar1)
? cMemoText                                // "Abcd\001efg\000ijk"
cMemoText  := CharRepl(CHR(0), @cMemoText, cNoChar2)
? cMemoText                                // "Abcd\001efg\002ijk"
```

**Classification:** add-on library, programming

**Compatibility:** Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:** FS2:CharList(), FS2:CharRem(), FS2:CharRepl(), FS2:CharOne(), FS2:CharSort(), MemoEncode(), MemoDecode()

# CHARNOT ()

---

**Syntax:**

`retC = CharNot ( expC1 )`

**Purpose:**

Complements each character in a string.

**Arguments:**

<expC1> is the character string that is processed. Embedded zero bytes in <expC1> are supported.

**Returns:**

<retC> is the modified string. Some of the resulting characters may contain zero byte chr(0) or chr(26) and if so, it is suggested to store <retC> in .dbt memo field by using MemoEncode(retC), see also LNG.4.2, FUN.MemoEncode() and CMD.REPLACE

**Description:**

CharNot() negates each bit in <expC>. This makes it possible to get a decreasing index sequence.

When a character string that has already been negated is negated again, the result is identical to the original.

**Example:**

```
? str := CharNot("123ABCabc")  
                                     // CHR(206,205,204,190,189,188,158,157,156)  
? CharNot(str)                       // "123ABCabc"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:CharAnd(), FS2:CharOr(), FS2:CharXor(), FS2:Complement(), Asc(), Chr(), Substr()



# CHARODD ()

---

**Syntax:**

```
retC = CharOdd ( expC1 )
```

**Purpose:**

Returns characters in the odd positions of a string.

**Arguments:**

<expC1> is the character string that is processed.

**Returns:**

<retC> is the modified string.

**Description:**

CharOdd() returns a string containing all the odd characters of <expC1>. It assembles all even characters within a string into a new string. The first position in a string is 1, and therefore odd.

Embedded zero bytes in <expC1> are not considered and hence terminates the string.

**Example:**

```
? CharEven("1234a")           // "24"
? CharEven(" H e l l o")       // "Hello"

? CharOdd("1234a")             // "13a"
? CharOdd(" H e l l o")        // " "
? CharOdd("H e l l o ")        // "Hello"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:CharEven(), FS2:CharMix()

# CHARONE ()

---

**Syntax:**

**retC = CharOne ( [expC1 ,] expC2 )**

**Purpose:**

Reduces adjoining duplicate characters in a string to one character.

**Arguments:**

<expC1> is a optional string specifying the characters that have their adjoining duplicates removed from <cString>. The default NIL value removes all adjoining duplicate characters.

<expC2> is the character string that is processed. If only one argument is given, it is assumed as <expC2>.

**Returns:**

<retC> is the modified string.

**Description:**

CharOne() searches within the <expC2> for repetitions of adjoining characters. When a character is removed, all of the characters but the first are deleted. This differs significantly from the CharList() function, where multiple occurrences of characters within the context of the string are removed.

Without the <expC1> parameter, all the repeating characters are removed. If the parameter is specified, only those characters in <expC1> are removed.

Embedded zero bytes in <expC1> are not considered and results in truncated <retC>.

**Example:**

```
? CharOne("122333a123")           // "123a123"
? CharOne(NIL, "122333a123")       // "123a123"
? CharOne("A B CCCD")              // "A B CD"
? CharOne(" ", "A B A B")          // "A B A B"
? CharOne("o", "122oooB12o")       // "122oB12o"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:WordOne(),      FS2:CharList(),      FS2:CharNolist(),      FS2:CharRem(),  
FS2:CharOnly()

# CHARONLY ()

---

**Syntax:**

**retC = CharOnly ( expC1, expC2 )**

**Purpose:**

Determines the common denominator between two strings on the basis of individual characters.

**Arguments:**

<expC1> is a string containing a sequence of characters that are not removed from <expC2>

<expC2> is the character string that is processed.

**Returns:**

<retC> is a string with the result.

**Description:**

CharOnly() removes all characters from <expC2> that are not in <expC1>. The function is particularly useful when comparing data that should have a standard format but has been input by different people. This category can consist of things like telephone numbers, part numbers, customer numbers, etc. (see example).

Embedded zero bytes in <expC1> or <expC2> are not considered and may result in truncated <retC>.

**Example:**

```
? CharOnly("0123456789", "213 - 39 07 923") // "2133907923"  
? CharOnly("0123456789", "213 / 390.79.23") // "2133907923"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:WordOnly(), FS2:CharRem(), FS2:CharOne(), FS2:CharList(), At()

# CHAROR ()

---

## Syntax:

```
retC = CharOr ( expC1, expC2 )
```

## Purpose:

Links the ASCII value(s) of string2 to all characters of string1 by using binary OR operation.

## Arguments:

<expC1> is the character string that is processed. Embedded zero bytes in <expC1> are supported.

<expC2> is the character string whose value is OR-ed to <expC1> Embedded zero bytes in <expC2> are supported.

## Returns:

<retC> is the modified string. Some of the resulting characters may contain zero byte chr(0) or chr(26) and if so, it is suggested to store <retC> in .dbt memo field by using MemoEncode(retC), see also LNG.4.2, FUN.MemoEncode() and CMD.REPLACE

## Description:

You can use CharOr() to produce simple character string coding, manipulation, set the high bit for all the characters in a string and so on.

CharOr() joins (links byte by byte using binary OR operation) each character in <expC1> with the corresponding character in <expC2>. When both character strings are the same length, then the first byte of <expC1> can be linked with the first byte of <expC2>, and the second byte of <expC1> can be linked with the second byte of <expC2>, and so on. If <expC2> is shorter than <expC1>, then as soon as the last byte of <expC2> is reached, the process continues and starts again with the first byte of <expC2>. However, if <expC1> is shorter than <expC2>, the process terminates with the end of <expC1>.

## Example:

```
? CharAnd("11111111", "123")           // "10110110"
? CharAnd("01234567", "1" )             // "01010101"
? CharAnd("01234xyz", "123")            // "00210012"
? CharAnd("123abc ", "123")              // "123!"# "

? charOr ("11111111", "123")             // "13313313"
? charOr ("01234567", "1" )              // "11335577"
? charOr ("01234xyz", "123")             // "13336{yz"
? charOr ("123abc ", "123")              // "123qrs12"

? CharOr (CHR(1) + CHR(2), "0")           // "12"
? charOr ("123ABCC[\]abc", CHR(32))       // "123abc(|)abc"
```

## Classification:

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:CharAnd(), FS2:CharAdd(), FS2:CharXor(), FS2:CharNot(), FS2:AddAscii(), Asc(), Chr(), Substr(), Strpoke()

# CHARPACK ()

---

## **Syntax:**

**retC = CharPack ( expC1, [expN2], [expL3] )**

## **Purpose:**

Compresses (packs) a string.

## **Arguments:**

<expC1> is the character string that is processed. Embedded zero bytes in <expC1> are supported.

## **Options:**

<expN2> is an optional numeric value specifying which pack algorithm is used. If not specified, 0 is the default.

= 0: Modified run length (RLL) encoding

= 1: Bit oriented algorithm (LZH)

<expL3> is an optional logical value. If

= .T. CharPack() ensures that the resulting string does not contain CHR(0) nor CHR(26) and the pack result can hence be safely stored in database .dbt memo fields. In worst case, the <retC> can be longer than <expC1> when this contain Chr(0) or Chr(26).

= .F. or not given: CharPack() ensures that the resulting string is not longer than the size of <expC1>, so the result can be safely stored in character database fields. In worst case, when the size would be longer, <retC> returns the input <expC1>.

## **Returns:**

<retC> is the modified, packed string according to the <expN2> and <expL3> switches.

## **Description:**

This function allows you to compress (pack) the contents of strings.

CharPack() can preferably be used to reduce the size of memo field in a database when data larger than 512 bytes are often used or to produce "crypted" results. Due of the unique check for the size, or avoiding of CHR(0) and CHR(26) in the result according to <expL3>, the results may be safely stored in a database.

Mode 0 uses modified "run-length encoding" (RLL). It is a kind of compression algorithm which replaces sequences ("runs") of consecutive repeated characters (or other units of data) with a single character and the length of the run. The longer and more frequent the runs are, the greater the compression that will be achieved. Run length encoding is actually not very useful for compressing text files, since a typical text file doesn't have a lot of long, repetitive character strings. It is very useful,

however, for compressing bytes of a monochrome image file, or data with many consecutive same characters.

Mode 1 uses modified "dynamic Lempel-Ziv-Huffman" compression (LZH) which is a variant of LZSS and LZW, one of the different Lempel-Ziv compression schemes. The algorithm relies on re-occurrence of byte sequences (tokens, strings) in its input. It maintains a table mapping input strings to their associated output codes, weighted by its relative frequency. It is very good suited for large text and binary data. Variants of LZW are used in GIF and TIFF image compression, and in the Unix compress or gzip command.

Since CT3 ignores the check/avoiding of memo field compatibility, the resulting <retC> is not compatible to CA3 Tools where the storage to memo fields often fails.

**Example:**

```
str := repli(" ",500) + "Hello world!" + repli(" ",500) // len =
1012
cPack1 := CharPack(@str, 0)
cPack2 := CharPack(@str, 0, .T.)
cPack3 := CharPack(@str, 1)
cPack4 := CharPack(@str, 1, .T.)
? len(cPack1),len(cPack2), len(cPack3),len(cPack4) // 27 27 69 70

str := "Hello world!" // len() = 12
cPack1 := CharPack(@str, 0)
cPack2 := CharPack(@str, 0, .T.)
cPack3 := CharPack(@str, 1)
cPack4 := CharPack(@str, 1, .T.)
? len(cPack1),len(cPack2), len(cPack3),len(cPack4) // 12 12 12 12

str := "Hello " + chr(26) + chr(0) + "world!" // len() = 14
cPack1 := CharPack(@str, 0)
cPack2 := CharPack(@str, 0, .T.)
cPack3 := CharPack(@str, 1)
cPack4 := CharPack(@str, 1, .T.)
? len(cPack1),len(cPack2), len(cPack3),len(cPack4) // 14 19 14 20

str := repli("Hello world!",50) // len() = 600
cPack1 := CharPack(@str, 0)
cPack2 := CharPack(@str, 0, .T.)
cPack3 := CharPack(@str, 1)
cPack4 := CharPack(@str, 1, .T.)
? len(cPack1),len(cPack2), len(cPack3),len(cPack4) // 600 600 132
133
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, call compatible to NT2/CA3 tools which supports only the first two arguments. Note: the resulting packed string is incompatible to Clipper, see text.

**Related:**

FS2:CharUnpack(), MemoEncode(), MemoDecode()

# CHARRELA ()

---

**Syntax:**

```
retN = CharRela ( expC1, expC2, expC3, expC4 )
```

**Purpose:**

Correlates the character positions in paired strings.

**Arguments:**

<expC1> is the string that is searched for in <expC2>.

<expC2> is a string that is processed by <expC1>.

<expC3> is the string that is searched for in <expC4>.

<expC4> is a string that is processed by <expC3>.

**Returns:**

<retN> is the positions where both <expC1> and <expC3> occur in the corresponding strings, or 0 when no relationship is found.

**Description:**

This function builds a relationship between two character strings. It determines the positions where the characters in <expC1> appear in <expC2> and where the characters in <expC3> appear in <expC4>.

Embedded zero bytes in <expC1> ... <expC4> are not considered and hence terminates the string.

**Example:**

```
// Search for the first common position in which a "b" appears in
// the first string and a "1" appears in the second:
? CharRela("b", "b b b b", "1", "bbb11111")      // 5
? CharRela("b", "1111bbb", "1", "bbb11111")      // 5
? CharRela("b", "1111bbb", "b", "bbb11111")      // 0
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:CharRelRep()



# CHARRELREP ()

---

**Syntax:**

```
retC = CharRelRep ( expC1, expC2, expC3, expC4,
                    expC5 )
```

**Purpose:**

Replaces characters in a string depending on their correlation.

**Arguments:**

<expC1> is the string that is searched for in <expC2>.

<expC2> is a string that is processed by <expC1>.

<expC3> is the string that is searched for in <expC4>.

<expC4> is a string that is processed by <expC3>.

<expC5> is one or more characters to replace those within <expC4> at the common position of <expC1> in <expC2> and <expC3> in <expC4>

**Returns:**

<retC> is the modified (or unchanged) string.

**Description:**

This function builds a relationship between two character strings. It determines the positions where the characters in <expC1> appear in <expC2> and where the characters in <expC3> appear in <expC4>. If such common positions are found, the replacement <expC5> replaces in <expC4> and the result returned in <retC>.

Embedded zero bytes in <expC1> ... <expC5> are not considered and hence terminates the string.

**Example:**

```
? CharRelRep("b", "b b b b", "1", "bbb11111", "x") // "bbb1x1x1"
? CharRelRep("b", "1111bbb", "1", "bbb11111", "x") // "bbb1xxx1"
? CharRelRep("b", "1111bbb", "b", "bbb11111", "x") // "bbb11111"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:CharRela()

# CHARREM ()

---

**Syntax:**

**retC = CharRem ( expC1, expC2 )**

**Purpose:**

Removes particular characters from a string.

**Arguments:**

<expC1> is/are the character(s) that is/are removed from <expC2>.

<expC2> is the character string that is processed.

**Returns:**

<retC> is the modified (or unchanged) string.

**Description:**

With this function you can remove particular characters from any position in <expC2>. It is possible to ensure that the character string does not contain these characters later.

Embedded zero bytes in <expC1> and <expC2> are not considered and hence terminates the string.

**Example:**

```
? CharRem(" ", " 1 2 1 ") // "121"
? CharRem("3y", "xyz123xyz3") // "xz12xz"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:CharOnly(), FS2:CharOne(), FS2:CharList()

# CHARREPL ()

---

**Syntax:**

**retC = CharRepl ( expC1, expC2, expC3, [expL4] )**

**Purpose:**

Replaces certain characters with others.

**Arguments:**

<expC1> is a list of characters to search for in <expC2>.

<expC1> is the character string that is processed.

<expC3> is the character list that replaces characters in <expC2>

**Options:**

<expL4> is an optional logical flag specifying whether multiple replacements are made. A .T. value designates one pass of the <expC1>, .F. (the default) multiple replacements.

**Returns:**

<retC> is the modified <expC2> string.

**Description:**

This function allows you to carry out very complex replacement procedures. Every character in the <expC1> is searched for within the <expC2>. If found, the character is replaced by the corresponding character in the <expC3>.

If <expL4> is not passed, characters are exchanged repeatedly as required. This means that the function goes through each individual character in the found characters in sequence, and then searches the entire <expC2>, exchanging <expC1> for corresponding characters in <expC3>. When you use this technique, characters that have already been replaced are exchanged again if the replacement character also appears in the search list.

However, if the optional <expL4> parameter is specified, the function proceeds differently. It goes through each character in <expC2> in sequence, determining whether or not it should be replaced. Characters that have been replaced are not replaced again.

As a rule, if the same characters appear within <expC3> and <expC1>, you must check very closely to determine which <expL4> parameter should be used (see example).

If the <expC3> sequence is shorter than <expC1>, the characters that do not have a corresponding replacement in <expC3> are replaced with the last character of <expC3> (see example).

Embedded zero bytes in <expC1> ... <expC3> are not considered and hence terminates the string.

**Example:**

```
// The number "1" is replaced with the letter "a", the number "2"
// with the letter "b", etc.. If the number "4" appeared in the
// character string, it would be replaced with a "d":

? CharRepl("1234", "1x2y3z", "abcd")           // "axbycz"

// The letters a-j are replaced with the numbers 0-9; for
// example, an "f" is replaced with a "6":

? CharRepl("abcdefghij", "jhfdb", "1234567890") // "08642"

// The third parameter makes fewer characters available for the
// exchange. Therefore the letters f-j are replaced with the last
// characters from "12345":

? CharRepl("abcdefghij", "jhfdb", "12345")      // "55542"

// Here is an example of the difference between a specified
// <expL4> parameter (.T.) and the default parameter (.F.):

? CharRepl("1234", "1234", "234A")             // "AAAA"
? CharRepl("1234", "1234", "234A", .T.)        // "234A"

// And here the analogy to the standard Strtran() function

? CharRepl("1", "12aB1211a", "x", .T.)         // "x2aBx2xxa"
? Strtran ("12aB1211a", "1", "x")              // "x2aBx2xxa"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:WordRepl(), FS2:WordToChar(), FS2:PosRepl(), FS2:CharRelRep(), Strtran()

# CHARSORT ()

---

**Syntax:**

```
retC = CharSort ( expC1, [expN2], [expN3], [expN4],  
                  [expN5], [expN6], [expL7] )
```

**Purpose:**

Sorts sequences within a string.

**Arguments:**

<expC1> is the character string that is processed.

**Options:**

<expN2> is an optional length of the sorting element. The default value is for 1 character. The maximal value is 255.

<expN3> is an optional number of characters that a sorting element takes into account in a comparison. The default value is <expN2> characters.

<expN4> is an optional number of characters at the beginning of the <expC1> that should not be taken into account in the sorting. The default value is 0.

<expN5> is an optional offset and designates from what position within the sorting element the comparison is made. The default value 0 is the first character.

<expN6> is an optional the length of the sort area relative to the <expN4> offset. The default value is 0.

<expL7> is an optional flag specifying whether the function sorts in ascending or descending order. If .F. or not specified, the function sorts in ascending order. When .T., the function sorts in descending order. The default value is .F. CharSort() accepts this flag at any last position of argument's list, see example.

**Returns:**

<retC> is the sorted <expC1> string or "" on error.

**Description:**

CharSort() allows you to sort the characters in a string in many different ways. Everything from the length of the sorting elements to the sort sequences is taken into account.

Embedded zero bytes in <expC1> are not considered and results in truncated <retC>.

**Example:**

```
// Sort in standard (ascending) and in descending order:  
? CharSort("Hello world!")                // " !Hde11loorw"  
? CharSort("Hello world!",,,,,.T.)         // "wroo11ledH! "  
? CharSort("Hello world!",.T.)             // "wroo11ledH! "  
  
// Sort standard, descending and in 2-byte chunks:
```

```

? CharSort("qwert")           // "eqrtw"
? CharSort("qwert", .T.)      // "wtrqe"
? CharSort("qwert", 2)        // "erqwt"

// Sort 2-byte length elements:

? CharSort("b1a4a3a1a2", 2)   // "a1a2a3a4b1"
? CharSort("b1a4a3a1a2", 2, .T.) // "b1a4a3a2a1"

// Sort 2-byte length elements, but only use the first or second
// character for the comparison:

? CharSort("b1a4a3a1a2", 2, 1) // "a4a3a1a2b1"
? CharSort("b1a4a3a1a2", 2, 1, 0, 1) // "b1a1a2a3a4"

// Sort individual characters, excluding the first three:

? CharSort("xxxqwert", 1, 1, 3) // "xxxeqrtw"
? CharSort("xxxqwert", 1, 1, 3, .T.) // "xxxwtrqe"

// Sort only the first four characters within a string:

? CharSort("384172852", 1, 1, 0, 0, 4) // "134872852"

```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

Asort(), At(), Asc()

# CHARSPREAD ()

---

**Syntax:**

```
retC = CharSpread ( expC1, expN2, [expCN3] )
```

**Purpose:**

Expands a string at separator positions.

**Arguments:**

<expC1> is the character string that is processed and expanded with spaces or <expCN3> characters to <expN2> length.

<expN2> is the length of the return string.

**Options:**

<expCN3> is an optional fill character and token delimiter. It can be either a character or a numeric value or a character (1..255). The default value is a space, CHR(32). If the <expCN3> is a string, only the first character is used. If the string is "", blank is used.

**Returns:**

<retC> is the modified, expanded string. If no <expCN3> separators are available in <expC1> or the string size is already longer than <expN2>, unmodified <expC1> is returned.

**Description:**

CharSpread() expands a string to a preset length. However, it behaves differently than the Expand() or PAdr() function. CharSpread() uses <expCN3> as the character to insert during the expansion. The string is expanded between tokens delimited by this character up to the length specified by <expN2>. This is similar to full justification in a text editor. When the delimiter does not appear in the string, no expansion takes place.

Previously existing fill characters <expCN3> are not taken into account in the redistribution (see examples). You also can call the CharOne() for the string before invoking CharSpread().

Embedded zero bytes in <expC1> are not considered and results in truncated <retC>.

**Example:**

```
? CharSpread("123456", 20, ".")           // "123456"
? CharSpread("1.2.3.4.5.6", 20, 46)        // "1...2...3..4...5...6"
? CharSpread("11..22..33", 20, ".")         // "11.....22.....33"
str := "11.22.....33"
? CharSpread(@str, 20, ".")                 // "11...22.....33"
? CharSpread(CharOne(".", @str), 20, ".")    // "11.....22.....33"
```

**Classification:** add-on library, programming

**Compatibility:** Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:** FS2:Expand(), FS2:CharOne(), PAdr(), Strtran()

# CHARSWAP ()

---

**Syntax:**

**retC = CharSwap ( expC1 )**

**Purpose:**

Exchanges all adjoining characters in a string.

**Arguments:**

<expC1> is the character string that is processed. Embedded zero bytes in <expC1> are supported.

**Returns:**

<retC> is the modified string.

**Description:**

CharSwap() takes all neighboring characters in <expC1> and exchanges them. As a result, bytes in even positions are exchanged for those in odd positions.

CharSwap() can be used for string integers generated by I2Bin() standard function that must be saved or sorted. The CharSwap() exchange must be carried out prior to calling CharSort() = low/high ordering of 16-bit integers, to achieve an accurate result.

**Example:**

```
? CharSwap("0123456789")           // "1032547698"

x1 := I2BIN(256)                     // bin: 00000000 00000001
x2 := I2BIN(1)                       // bin: 00000001 00000000
s1 := CharSwap(x1)                   // bin: 00000001 00000000
s2 := CharSwap(x2)                   // bin: 00000000 00000001
? strpeek(x1,1), strpeek(x1,2), "->", ;
  strpeek(s1,1), strpeek(s1,2)       // "0x0 0x1" -> "0x1 0x0"
? strpeek(x2,1), strpeek(x2,2), "->", ;
  strpeek(s2,1), strpeek(s2,2)       // "0x1 0x0" -> "0x0 0x1"

? x2 > x2, "->", s1 > s2             //      .F.      ->      .T.
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:WordSwap(), FS2:CharSort()



# CHARUNPACK ()

---

**Syntax:**

**retC = CharUnpack ( expC1 )**

**Purpose:**

Decompresses (unpacks) a string compressed by CharPack().

**Arguments:**

<expC1> is the character string that is processed. Embedded zero bytes in <expC1> are supported.

**Returns:**

<retC> is the decompressed or unmodified string.

**Description:**

This function unpacks strings compressed by the CharPack() function. The used pack method is automatically recognized according to the packed string prefix. If the string could not be packed (in view of <expL3> setting at CharPack() processing), or was not packed by FS2 Toolbox, unmodified string is returned.

**Example:**

```
str := replicate("x",500) + "Hello world!" + repli(" ",500)
cPack := CharPack(@str, 0)
cUnpk := CharUnpack(@cPack)
? len(str), len(cPack), str == cunpk    // 1012 27 .T.

str := replicate("Hello " + chr(26) + chr(0) + "world!", 100)
cPack := CharPack(@str, 1, .T.)
cUnpk := CharUnpack(@cPack)
? len(str), len(cPack), str == cunpk    // 1400 220 .T.
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, call compatible to NT2/CA3 tools. Note: the resulting packed string from CharPack() is incompatible to Clipper, see details in CharPack(). For that reason CharUnpack() of FS2 Tools cannot process CT3 packed strings and vice versa.

**Related:**

FS2:CharPack(), MemoEncode(), MemoDecode()

# CHARXOR ()

---

## Syntax:

```
retC = CharXor ( expC1, expC2 )
```

## Purpose:

Links the ASCII value(s) of string2 to all characters of string1 by using binary XOR operation.

## Arguments:

<expC1> is the character string that is processed. Embedded zero bytes in <expC1> are supported.

<expC2> is the character string whose value is XOR-ed to <expC1>. Embedded zero bytes in <expC2> are supported.

## Returns:

<retC> is the modified string. Some of the resulting characters may contain zero byte chr(0) or chr(26) and if so, it is suggested to store <retC> in .dbt memo field by using MemoEncode(retC), see also LNG.4.2, FUN.MemoEncode() and CMD.REPLACE

## Description:

You can use CharXor() to produce "exclusive OR" operation on each character in <expC1> with the corresponding characters in <expC2>. Exclusive OR is often used to crypt text (like passwords or important database fields) by using a secret "key". See also Crypt().

The crypting by CharXor() is reversive when the same key is used.

CharXor() joins (links byte by byte using binary XOR operation) each character in <expC1> with the corresponding character in <expC2>. When both character strings are the same length, then the first byte of <expC1> can be linked with the first byte of <expC2>, and the second byte of <expC1> can be linked with the second byte of <expC2>, and so on. If <expC2> is shorter than <expC1>, then as soon as the last byte of <expC2> is reached, the process continues and starts again with the first byte of <expC2>. However, if <expC1> is shorter than <expC2>, the process terminates with the end of <expC1>.

## Example:

```
cCoded := CharXor("Hello world!", "myPassword")
? CharXor(cCoded, "myPassword")           // "Hello world!"
? CharXor(cCoded, "Mypassword")           // "heLlo world!"

cCoded := CharXor("11111111", "1")         // 8 * CHR(0)
? CharXor(cCoded, "1")                    // "11111111"
? CharXor(cCoded, "2")                    // "22222222"
```

**Example:** The user's name and password are stored in database character fields ('user' and 'passw' ea C 10), both are case sensitive and crypted. The password's secret key (expC2) is a combination of user name and fixed string.

```
cUser := cPassw := space(10)    // should be same as field length
@ 1,0 SAY "Username" GET cUser      valid !empty(cUser)
@ 2,0 SAY "Password" GET cPassw PICT "@P" valid !empty(cPassw)
READ

cCryptUser := CharXor(cUser, "My Secrets")
cCryptPass := CharXor(cPassw, CharXor(cUser, "My Secrets"))

USE myusers
locate for myusers->user == cCryptUser
if !found()
    ok := alert("User not available. Create new?", {"yes","no"})
    if ok != 1
        quit
    endif
    APPEND BLANK
    REPLACE myusers->user with cCryptUser
    REPLACE myusers->passw with cCryptPass
endif

if !(cCryptPass == myusers->passw)
    alert("sorry, wrong password...")
    quit
endif

// or:
cDecrUser := CharXor(myusers->user, "My Secrets")
cDecrPass := CharXor(myusers->passw, ;
                    CharXor(cDecrUser, "My Secrets") )
if(cPassw != cDecrPass)
    alert("sorry, wrong password...")
    quit
endif
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:CharAnd(), FS2:CharAdd(), FS2:CharOr(), FS2:CharNot(), FS2:Crypt(),  
FS2:AddAscii(), Asc(), Chr(), Substr(), Strpoke()

# CHECKSUM ()

---

**Syntax:**

```
retN = CheckSum ( expC1 )
```

**Purpose:**

Calculates the checksum for a character string using an algorithm.

**Arguments:**

<expC1> is the character string that is processed. Embedded zero bytes in <expC1> are supported.

**Returns:**

<retN> is the checksum.

**Description:**

CheckSum() calculates the checksum for a character string. This checksum can determine if a character string has been changed, or transmitted or typed incorrectly.

Note: The FS2 Tools CheckSum() intentionally use the same algorithm as CA-Tools3, which says: "The CheckSum() cannot be used for definitive coding of data because the calculated sum for two different character strings could be the same. For example, the checksum for 'PASTETEN' and 'PERSONAL' are identical."

For that reason, best to use Crc16() or Crc32() instead, which does not return the same checksum for different strings.

**Example:**

```
? CheckSum("abc")           // 50382374
? CheckSum("cba")           // 50381862
? CheckSum("PASTETEN")      // 134354532
? CheckSum("PERSONAL")     // 134354532
? CheckSum("")              // 0
? CheckSum("abc"+chr(0)+"xyz") // 117584529

? crc16("abc")              // 38712
? crc16("cba")              // 38424
? crc16("PASTETEN")         // 61327
? crc16("PERSONAL")         // 8415
? crc16("")                 // 0
? crc16("abc" + chr(0) + "xyz") // 59038

? crc32("abc")              // 891568578
? crc32("cba")              // -659622784
? crc32("PASTETEN")         // 1323612292
? crc32("PERSONAL")         // 934932936
? crc32("")                 // 0
? crc32("abc" + chr(0) + "xyz") // 1070356630
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:AsciiSum(), FS2:Crc16(), FS2:Crc32()

# COMPLEMENT ()

---

**Syntax:**

```
ret = Complement ( exp1 )
```

**Purpose:**

Forms the complement value of any scalar data type.

**Arguments:**

<exp1> is expression of any data type to form a complement value.

**Returns:**

<ret> is the result of the Complement() function. The data type is the same as the parameter.

**Description:**

This Complement() function returns the respective opposite value of the <exp1> parameter. The result is reversible by a new Complement() call.

valtype(exp1)	<ret> value
C = character	CharNot(exp1)
N = numeric	exp1 * -1.0
L = logical	!exp1
D = date	(01/01/3000 - exp1) but as date type
U = NIL	NIL
A = array	exp1 (unchanged)
B = code block	exp1 (unchanged)
O = object	exp1 (unchanged)
other	exp1 (unchanged)

**Example:**

```
SET CENTURY ON
? Complement(.T.)           // .F.
? Complement(.F.)           // .T.
? Complement(99)             // -99.00
? Complement(0)              // 0
? Complement(-9.9)           // 9.90
? Complement("12ABab")      // CHR(206,205,190,189,158,157)
? Complement(CTOD("/ / "))  // 01/01/3000
? Complement(CTOD("05/28/2002")) // 08/06/0998
? Complement(CTOD("08/06/0998")) // 05/28/2002
? Complement()              // NIL
? Complement( {1,2} )       // array {1,2}
```

**Classification:** add-on library, programming

**Compatibility:** Available in FS2-Toolbox, compatible to NT2/CA3 tools. The date type complement may differ, since CT3 does not consider leap years.

**Related:** FS2:CharNot(), FS2:Blank()

# COUNTLEFT ()

---

**Syntax:**

**retN = CountLeft ( expC1, [expCN2] )**

**Purpose:**

Counts a particular character at the beginning of a string.

**Arguments:**

<expC1> is the character string that is processed. Embedded zero bytes in <expC1> are supported.

**Options:**

<expCN2> is either a character or its numeric equivalence which occurrence is counted at the beginning of <expC1>. If not specified, the default is a space = chr(32). If the <expCN2> string is longer, only the first character is used.

**Returns:**

<retN> is the number of occurrences of <expCN2> at the beginning of <expC1>

**Description:**

CountLeft() is similar to RemLeft(). While RemLeft() removes leading characters from the <expC1>, CountLeft() only determines the number of leading <expCN2> characters appearing in an uninterrupted sequence at the beginning of the <expC1>.

**Example:**

```
? CountLeft(" 123")           // 3
? CountLeft("..4.123", ".")    // 2
? CountLeft("123456")         // 0
str := replicate(chr(0),5) + "xyz"
? CountLeft(@str, chr(0))      // 5
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:CountRight(), FS2:RemLeft()

# COUNTRIGHT ()

---

**Syntax:**

`retN = CountRight ( expC1, [expCN2] )`

**Purpose:**

Counts a particular character at the end of a string.

**Arguments:**

<expC1> is the character string that is processed. Embedded zero bytes in <expC1> are supported.

**Options:**

<expCN2> is either a character or its numeric equivalence which occurrence is counted at the end of <expC1>. If not specified, the default is a space = chr(32). If the <expCN2> string is longer, only the first character is used.

**Returns:**

<retN> is the number of occurrences of <expCN2> at the end of <expC1>

**Description:**

CountRight() is similar to RemRight(). While RemRight() removes trailing characters from the <expC1>, CountRight() only determines the number of trailing <expCN2> characters appearing in an uninterrupted sequence at the end of the <expC1>.

**Example:**

```
? CountRight("abc ")           // 3
? CountRight("abc.d..", ".")    // 2
? CountRight("123456")          // 0
str := "abc" + replicate(chr(0),5)
? CountRight(@str, chr(0))       // 5
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:CountLeft(), FS2:RemRight()



# CRC16 ()

---

## Syntax:

```
retN = Crc16 ( expC1, [expN2], [expN3] )
```

## Purpose:

Computes a Cyclic Redundancy Check (CRC) for the string, conformed to ANSI X3.66, ADCCP, CCITT X.25

## Arguments:

<expC1> is the character string that is processed. Embedded zero bytes in <expC1> are supported.

## Options:

<expN2> is an optional starting value that is added to the CRC. The default value is 0.

<expN3> is the used polynomial:

```
= 40961 and <expN2> = 0 : XMODEM compatible CRC (default)
= 4129 and <expN2> = 0 : X25 compatible CRC
= 4129 and <expN2> = 65535 : CCITT compatible CRC
= 32773 and <expN2> = 0 : ARC compatible CRC
```

## Returns:

<retN> is the CRC value as 16-bit value.

## Description:

The Cyclic Redundancy Check (CRC) can be used for validity checking and for security purposes and for test of communication errors. It uses 16-bit CRC value conforming to ANSI X3.66, ADCCP, CCITT X.25

Note: you may use the standard Crc32() function if you wish to increase the precision to a 32 bit value.

## Example:

```
? crc16("abc") // 38712
? crc16("cba") // 38424
? crc16("PASTETEN") // 61327
? crc16("PERSONAL") // 8415
? crc16("") // 0
? crc16("abc" + chr(0) + "xyz") // 59038

? Com_crc("abc") // 38712
? Com_crc("cba") // 38424
? Com_crc("PASTETEN") // 61327
? Com_crc("PERSONAL") // 8415
? Com_crc("") // 0
? Com_crc("abc" + chr(0) + "xyz") // 59038

? crc32("abc") // 891568578
? crc32("cba") // -659622784
? crc32("PASTETEN") // 1323612292
? crc32("PERSONAL") // 934932936
```

```
? crc32("") // 0
? crc32("abc" + chr(0) + "xyz") // 1070356630
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, not available in NT2/CA3 tools.

**Related:**

Crc32(), FS2:Com\_crc(), FS2:Checksum()

# CRYPT ()

---

**Syntax:**

```
retC = Crypt ( expC1, [expC2] )
```

**Purpose:**

Encrypts and decrypts a string.

**Arguments:**

<expC1> is the character string that is processed.

<expC2> is a "key" or "password" used for crypting of <expC1>. If not given, an internal password is used. However, it is suggested always to use your own password, to ensure security.

**Returns:**

<retC> is the crypted or decrypted string or "" on error. Crypt() ensures that zero byte chr(0) nor chr(26) are included in <retC> so the crypted result may be safely stored in database fields and .dbt memo fields. When storing the crypted string into database field, make sure the field size is at least 2 bytes larger than the original (uncrypted) string, safe is double field size.

**Description:**

In contrast to a simple crypt method using CharXor(), this function has a random number generator that uses a "random seed" algorithm. This makes the encrypting even more secure.

Encrypted strings can be decrypted with the same password. With multiple encryptions, the decryption must occur in reverse order.

Note: the password should be as long as possible - common is a minimum of six characters, preferably more.

Embedded zero bytes in <expC1> and <expC2> are not considered and may result in truncated <retC>.

**Technical note:**

The returned length of crypted string is at least 2 bytes larger than the length of the original string, but in some condition it may result in twice length of original size. Note that de-crypt algorithm assumes correct length of crypted string, i.e. you may need to use trim() or left() when the crypted data is stored in database character field (= fixed length padded by spaces). In .dbt (memo) and .dbv fields of variable length, and in character variables, the crypted string length is detected automatically. For character fields, using the CharXor() instead of Crypt() may be better alternative, since it does not change the string length.

**Example:**

```
cSecret := Crypt("Hello world!", "myPassw")
? len(cSecret)           // 14
? Crypt(cSecret, "myPassw") // "Hello world!"
? Crypt(cSecret, "Mypassw") // "Jejlo w rjd!"
```

**Example:** The user's password is stored in database character field (C 20)

in format <n><crypt><spaces> where <n> is the size of the crypted string and <spaces> are filled automatically. Note: using <n> avoids headaches when changing the database structure later :-) The secret key here is a combination of user name and fixed string. See also similar example in CharXor() which is less "complicated".

```
cUser := cPassw := space(10)
@ 1,0 SAY "Username" GET cUser          valid !empty(cUser)
@ 2,0 SAY "Password" GET cPassw PICT "@P" valid !empty(cPassw)
READ

USE myusers
cCrypt := crypt(cPassw, CharXor(cUser, "My Secrets"))
cCrypt := padr(chr(len(cCrypt))      ; /* <n>      here: 12 */
               + cCrypt,              ; /* <crypt>  crypted PW */
               len(myusers->passw) )  ; /* <spaces> to fill */

locate for myusers->user == cUser      // 'user' field = c 10
if !found()
  ok := alert("User not available. Create new?", {"yes","no"})
  if ok != 1
    quit
  endif
  APPEND BLANK
  REPLACE myusers->user with cUser
  REPLACE myusers->passw with cCrypt
endif

if !(cCrypt == myusers->passw)
  alert("sorry, wrong password...")
  quit
endif

// or:
cDecrypted := crypt( substr(myusers->passw, 2, ;
                           asc(left(myusers->passw,1)) ), ;
                    CharXor(cUser,"My Secrets") )
if(cPassw != cDecrypted)
  alert("sorry, wrong password...")
  quit
endif
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. Due of the hidden algorithm, the results of FS2 Crypt() are incompatible to those of CT3.

**Related:**

FS2:CharXor()

# CSETATMUPA ()

---

**Syntax:**

**retL** = CsetAtMupa ( [expL1] )

**Purpose:**

Determines the setting of the multi-pass mode for AT\*() functions.

**Options:**

<expL1> is a logical value specifying the multi-pass mode for all AT\*() functions. The default value .F. sets off the multi-pass mode. If <expL1> is not given, only the current setting is returned.

**Returns:**

<retL> is the current multi-pass mode setting at the time of entering this function.

**Description:**

CsetAtMupa() sets a switch, that affects the operation of several FS2 Tools functions. These functions permit targeted, extremely flexible substring manipulation. The affected functions are:

ATNUM()	AFTERATNUM()	BEFORATNUM()
ATREPL()	NUMAT()	ATADJUST()
WORDTOCHAR()	WORDREPL()	

**Example:**

```
? CsetAtMupa()           // .F.  
? CsetAtMupa(.T.)       // .F.  
? CsetAtMupa()           // .T.
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:AtNum(), FS2:AfterAtNum(), FS2:AtRepl(), FS2:AtAdjust(), FS2:BeforAtNum(), FS2:NumAt(), FS2:WordToChar(), FS2:WordRepl()

# CSETREF ()

---

## Syntax:

**retC = CsetRef ( [expL1] )**

## Purpose:

Determines whether or not reference sensitive functions return a value (in CA-Tools3 only).

## Options:

<expL1> is a logical value specifying (in CA-Tools3) if the function return value should be suppressed and the incoming value, passed by reference should be modified instead.

## Returns:

<retL> is the status of CsetRef() at the time of entering this function.

## Description:

In CA-Tools3, this setting allows to directly modify the incoming parameters by the returned value. FS2 Toolbox does not support this "feature". Upon mature consideration we decided that changing incoming arguments is an extremely danger and usually bad programming technique, causing many headaches at the time of developing and debugging of YOUR application.

Therefore, the FS2 functions will never modify the incoming values (except when explicitly notified, e.g. in the StrSwap() function). For that reason, the status of CsetRef() is ignored in FS2. Use the corresponding function return value instead.

You may, however, pass the argument variables (parameter) by reference, i.e. by prefixing the variable by the "@" sign (see LNG.2.3.2), the execution will then work slightly faster.

To make your porting of old Clipper application easier, CsetRef(.T.) will display run-time warning at the corresponding source position when FS\_SET("devel",.T.) is set. You also may use 'grep -in csetref \*.prg' to detect it use before re-compiling.

## Example:

```
CsetRef(.F.)
? cStr1 := "original"           // "original"
? cStr2 := CharSort(cStr1)      // "agiilnor" = CT3: "agiilnor"
? cStr1                          // "original" = CT3: "original"

? cStr2 := CharSort(@cStr1)     // "agiilnor" = CT3: "agiilnor"
? cStr1                        // "original"   CT3: "agiilnor" (!)

CsetRef(.T.)
? cStr1 := "original"           // "original"
? cStr2 := CharSort(@cStr1)     // "agiilnor"   CT3: .F.           (!)
? cStr1                        // "original"   CT3: "agiilnor" (!)
```

**Classification:**

add-on library, programming

**Compatibility:**

Available but ignored in FS2-Toolbox, call-compatible to CA3 tools.

**Related:**

FS\_SET("devel"), LNG.2.3.2

# EXPAND ()

---

**Syntax:**

**retC = Expand ( expC1, [expN2], [expCN3] )**

**Purpose:**

Expands a string by inserting characters.

**Arguments:**

<expC1> is the character string that is processed. Embedded zero bytes in <expC1> are supported.

**Options:**

<expN2> is the number of the <expCN2> characters that are inserted between each character in the <expC1>. If not specified, the default is 1. If <expN2> is a character, it is assumed as <expCN3>.

<expCN1> is the character that is inserted between each character in <expC1>. It can be either a character or it numeric value in range (1..255). The default value is a space = CHR(32). If the <expCN3> is a string, only the first character is used. If the string is "", blank is used.

**Returns:**

<retC> is the modified (or unchanged) string.

**Description:**

Expand() can be used to justify text output. The expansion only takes place when the <expC1> contains at least two characters.

To fill out a character string at the ends, use the Padr() or FS2:PadRight() functions. To insert characters at the text begin, use Padl() or FS2:PadLeft() functions. To fill characters at both text begin and end, use Padc() or FS2:Center() functions.

**Example:**

```
? Expand("123456")           // "1 2 3 4 5 6"
? Expand("123456", 2)        // "1 2 3 4 5 6"
? Expand("123456", ".")      // "1.2.3.4.5.6"
? Expand("123456", 2, ".")    // "1..2..3..4..5..6"
? Expand("123"+chr(0)+"45", 2, ".") // "1..2..3.." + CHR(0) + "..4..5"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:CharSpread(), FS2:PadLeft(), FS2:PadRight(), FS2:Center, Padl(), Padr(), Padc(), Strtran()



# HEXTOSTR ()

---

**Syntax:**

```
retC = HexToStr ( expC1 )
```

**Purpose:**

Converts a sting containing hexadecimal values into byte sequence.

**Arguments:**

<expC1> is a string where each two characters represents one hex character returned in <retC>. The <expC1> can only contain characters "0" to "9", "A" to "F" and "a" to "f". When the length of <expC1> is not even, a "0" character is assumed at the begin of <expC1>.

**Returns:**

<retC> are characters converted from the <expC1> hex representation, or "" on error.

**Description:**

HexToStr() returns a character string that contains the ASCII representation of the <expC1> hex string. The returned size is half as long as <expC1>.

The reverse function is StrToHex(). For converting hex string into numeric value, use the standard FlagShip function Hex2num().

**Example:**

```
? HexToStr("31415a")           // "1AZ"
? HexToStr("31415")             // chr(3) + chr(20) + chr(21)
? HexToStr("031415")            // chr(3) + chr(20) + chr(21)
? HexToStr("Ab3f")              // chr(171) + chr(63)
? HexToStr("F")                 // chr(15)
? HexToStr(" x")                // ""      (error, illegal characters)
? HexToStr("")                  // ""
? StrToHex("Xn ")               // "586E20"
? HexToStr("586e20")            // "Xn "
? Hex2num("586e20")             // 5795360
? Hex2num("0x586e20")           // 5795360
? HexToStr(Num2hex(5795360))    // "Xn "
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:Hex2Str(), Hex2Num(), Num2Hex()

# JUSTLEFT ()

---

**Syntax:**

`retC = JustLeft ( expC1, [expCN2] )`

**Purpose:**

Moves characters from the beginning to the end of a string.

**Arguments:**

<expC1> is the character string that is processed. Embedded zero bytes in <expC1> are supported.

**Options:**

<expCN2> is either a character or its numeric equivalence which should be moved from begin to the end of <expC1>. If not specified, the default is a space = chr(32). If the <expCN2> string is longer, only the first character is used.

**Returns:**

<retC> is the modified string.

**Description:**

JustLeft() moves the characters specified in <expCN2> from the beginning of the <expC1> string to the end. Thereafter the remaining text in the <expC1> string is left justified, without affecting the length.

**Example:**

```
? JustLeft(" 123")           // "123  "  
? JustLeft("..123", ".")     // "123.."  
? JustLeft("..123" + chr(0) + "45", ".") // "123"+chr(0)+"45.."
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:JustRight(), FS2:PadRight(), Padr()

# JUSTRIGHT ()

---

**Syntax:**

```
retC = JustRight ( expC1, [expCN2] )
```

**Purpose:**

Moves characters from the end a string to the beginning.

**Arguments:**

<expC1> is the character string that is processed. Embedded zero bytes in <expC1> are supported.

**Options:**

<expCN2> is either a character or it numeric equivalence which should be moved from the end to begin of <expC1>. If not specified, the default is a space = chr(32). If the <expCN2> string is longer, only the first character is used.

**Returns:**

<retC> is the modified string.

**Description:**

JustLeft() moves the characters specified in <expCN2> from the end of the <expC1> string to the beginning. Thereafter the remaining text in the <expC1> string is right justified, without affecting the length.

**Example:**

```
? JustRight("123 ")           // " 123"
? JustRight("123..",".")      // "..123"
? JustRight("123" + chr(0) + "45..",".") // "..123"+chr(0)+"45"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:JustLeft(), FS2:PadLeft(), Padl()

# LIKE ()

---

## Syntax:

**retL = Like ( expC1, expC2 )**

## Purpose:

Compares character strings using wildcard characters.

## Arguments:

<expC1> the wildcard string, or a "mask" for comparison. All wildcards must be in this character string

<expC2> is the character string that is processed/compared.

## Returns:

<retL> is .T. if the <expC1> mask match <expC2>, or .F. if not.

## Description:

Like() allows you to compare two strings with one another, where the first can contain wildcard characters. This is similar to the way wildcard characters are used in conjunction with OS commands but not fully identical. A "?" in <expC1> matches one corresponding character in <expC2>. A "\*" in <expC1> matches zero, one or more characters in <expC2>.

Embedded zero bytes in <expC1> or <expC2> are not considered.

## Example:

```
? Like("XYZ?", "XYZ")           // .F.
? Like("XYZ?", "XYZ1")          // .T.
? Like("*OG.*", "PROG.PRG")     // .T.
? Like("*OG.*", "LOG.PRG")      // .T.

? Like("*R*T*", "PROT")          // .T.
? Like("*R*T*", "xRT")          // .T.
? Like("*R*T*", "PROTO")        // .T.
? Like("*R*T*?", "PROTO")       // .T.
? Like("*R*T*?", "PROTO2")      // .T.

? Like("*llo", "Hello")         // .T.
? Like("*llo", "HELLO")         // .F.
? Like("*llo", "Hallo")        // .T.
? Like("Hello", "*llo")         // .F.
```

## Classification:

add-on library, programming

## Compatibility:

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only the first optional argument.

# LTOC ()

---

**Syntax:**

**retC** = LtoC ( [expL1] )

**Purpose:**

Converts a logical value into a character.

**Options:**

<expL1> is the logical value that is converted to string. If not specified, .F. is assumed.

**Returns:**

<retC> is a string corresponds to the logical value specified in the <expL1> parameter.

**Description:**

In contrast to the LtoN() function, which converts a logical value into a number, LtoC() returns corresponding string. This can be particularly helpful with combined index keys or for concatenating character values with logicals.

**Example:**

```
? LtoC()           // "F"
? LtoC(.F.)        // "F"
? LtoC(.T.)        // "T"
? LtoC(Year(Date()) > 2500) // "F"
? LtoC(DELETED())  // "T" or "F"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:LtoN()

# MAXLINE ()

---

**Syntax:**

**retN = MaxLine ( expC1 )**

**Purpose:**

Finds the longest line within a string which includes CR or LF.

**Arguments:**

<expC1> is the character string that is processed. This is usually a string edited by MemoEdit().

**Returns:**

<retN> is the longest line (substring of <expC1>) between the begin, end or two CR = chr(13) or LF = chr(10)

**Description:**

MaxLine() looks for the longest line within a character string. This allows you to determine the required width for a window to display the text via MemoEdit() without additional line breaks.

With the exception of the carriage return (CR) and line feed (LF), MaxLine() treats all characters as printable with a width of 1. If the <expC1> text contains tabs, then you must first use the TabExpand() function to expand the tabs to spaces.

Embedded zero bytes in <expC1> are not considered and hence terminates the string.

**Example:**

```
str := "Hello" + chr(10) + "world, that's me."
str2 := chr(10) + chr(10) + repli("x",40) + chr(10) + "aaa"

? MaxLine(str)           // 17
? MaxLine(str + str2)    // 40
? NumLine(str)           // 2
? NumLine(str + str2)    // 5
wait

cNew := MyMemoDispl(str)    // edit/display the string
cNew := MyMemoDispl(str + str2) // edit/display the string

FUNCTION MyMemoDispl(cText)
    local iRows, iCols

    iRows := minmax( 4, NumLine(cText), MaxRow() -1)
    iCols := minmax(20, MaxLine(cText), MaxCol() -1)

    @ 0, 0 TO iRows +1, iCols +1 DOUBLE
    return MemoEdit(cText, 1, 1, iRows, iCols, .F.)
```

***Classification:***

add-on library, programming

***Compatibility:***

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

***Related:***

FS2:NumLine(), FS2:TabExpand(), MemoEdit(), MemoLine()

# NUMAT ()

---

**Syntax:**

**retN = NumAt ( expC1, expC2, [expN3] )**

**Purpose:**

Counts the number of occurrences of a sequence within a string.

**Arguments:**

<expC1> is a string that is searched in <expC2>.

<expC2> is the string that is processed, i.e. searched thru.

**Options:**

<expN3> is the number of characters that are excluded from the search. The default value 0 ignores none.

**Returns:**

<retN> is a count specifying how often the <expC1> was found in the <expC2> string.

**Description:**

NumAt() determines how often a particular <expC1> appears within <expC2>. When you use <expN3> you can skip the specified number of characters at the beginning of the <expC2>.

The setting for CsetAtMupa() impacts your results. The string is searched from the left for each occurrence of the <expC1> string. If CsetAtMupa() is .F. (the default), then the search continues after the last character of the found sequence. If CsetAtMupa() is .T., then the search continues after the first character of the found sequence.

Embedded zero bytes in <expC1> or <expC2> are not considered.

**Example:**

```
? NumAt("ab", "abcdeabc")           // 2
? NumAt("ab", "abcdeabc", 1)         // 1

? CsetAtMupa()                       // .F.
? NumAt("aa", "aaaab")               // 2
? CsetAtMupa(.T.)                   // .F.
? CsetAtMupa()                       // .T.
? NumAt("aa", "aaaab")               // 3
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:CsetAtMupa(), At()



# NUMLINE ()

---

**Syntax:**

**retN = NumLine ( expC1, [expN2] )**

**Purpose:**

Counts the number of lines within a string which includes CR or LF.

**Arguments:**

<expC1> is the character string that is processed. This is usually a string edited by MemoEdit().

**Options:**

<expN2> is an optional max length of the output line when LF was not yet found. The default value is 80.

**Returns:**

<retN> is the number of lines in <expC1> counted between the begin, CR = chr(13) an/or LF = chr(10) and the end of the string.

**Description:**

NumLine() looks for the number of lines within a character string. This allows you to determine the required height for a window to display the text via MemoEdit() without additional line breaks.

With the exception of the carriage return (CR) and line feed (LF), NumLine() treats all characters as printable with a width of 1. If the <expC1> text contains tabs, then you must first use the TabExpand() function to expand the tabs to spaces.

Embedded zero bytes in <expC1> are not considered and hence terminates the string.

**Example:**

see example in MaxLine()

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:MaxLine(), FS2:TabExpand(), MemoEdit(), MemoLine()

# NUMTOKEN ()

---

**Syntax:**

**retN = NumToken ( expC1, [expC2], [expN3] )**

**Purpose:**

Determines the number of tokens in a string.

**Arguments:**

<expC1> is the character string that is processed. Embedded zero bytes are supported.

**Options:**

<expC2> is a list of delimiters used to identify the tokens. If not specified, the default <expC2> delimiters are:

" ,. ; : ! ? \ / < > ( ) # ^ % & + - \* " + chr(9) + chr(13) + chr(10) + chr(26) + chr(138) + chr(141) + chr(4) + chr(0)

Embedded zero bytes are supported.

<expN3> is a max number of delimiters before empty token is returned. The default is 0 indicating that empty tokens are not taken into account.

**Returns:**

<retN> is the number of token in <expC1> string.

**Description:**

NumToken() determines how many words (or tokens) are contained in the string <expC1>.

**Example:**

```
? NumToken("Good Morning!") // 2
cString := "Yes! That's it. Maybe not?"
? NumToken(cString, ".! ?") // 3
cString := "one,two,,four"
? NumToken(cString, ", ") // 3
? NumToken(cString, " ", 0) // 3
? NumToken(cString, " ", 1) // 4
? NumToken(cString, " ", 2) // 3
? NumToken("Have a nice day.") // 4
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:Token(), FS2:AtToken(), FS2:TokenArray()

# PADLEFT ()

---

**Syntax:**

```
retC = PadLeft ( expC1, expN2, [expCN3] )
```

**Purpose:**

Pads a string on the left to a particular length.

**Arguments:**

<expC1> is the character string that is processed. Embedded zero bytes in <expC1> are supported.

<expN2> is the new length for <expC1>

**Options:**

<expNC3> is either a character or it numeric equivalence which should be filled at the begin of <expC1>. If not specified, the default is a space = chr(32). If the <expCN2> string is longer, only the first character is used.

**Returns:**

<retC> is the modified string.

**Description:**

PadLeft() work same as the standard function Padl(). The only difference is in truncating of the <expC1> when <expN2> is shorter than the length of <expC1>: While Padl() leaves the begin of the string, i.e. work like LEFT(expC1,expN2), the PadLeft() leaves the right reminder of the string, i.e. work like RIGHT(expC1,expN2)

**Example:**

```
? PadLeft("123456", 4)           // "3456"
? PadLeft("123456", 8)           // " 123456"
? PadLeft("123456", 8, ".")      // "..123456"
? PadLeft("123456", 8, 43)       // "++123456"
? PadLeft("123"+chr(0)+"456", 8)  // " 123"+chr(0)+"456"

? Padl("123456", 4)              // "1234"
? Padl("123456", 8)              // " 123456"
? Padl("123456", 8, ".")         // "..123456"
? Padl("123"+chr(0)+"456", 8)    // " 123"+chr(0)+"456"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:PadRight(), Padl(), Padr(), Padc()

# PADRIGHT ()

---

**Syntax:**

```
retC = PadRight ( expC1, expN2, [expCN3] )
```

**Purpose:**

Pads a string on the right to a particular length.

**Arguments:**

<expC1> is the character string that is processed. Embedded zero bytes in <expC1> are supported.

<expN2> is the new length for <expC1>

**Options:**

<expNC3> is either a character or its numeric equivalence which should be filled at the begin of <expC1>. If not specified, the default is a space = chr(32). If the <expCN2> string is longer, only the first character is used.

**Returns:**

<retC> is the modified string.

**Description:**

PadRight() works the same as the standard function PAdr(). Both truncate the <expC1> when <expN2> is shorter than the length of <expC1> leaving the begin of the string, i.e. work like LEFT(expC1,expN2) in such a case.

**Example:**

```
? PadRight("123456", 4)           // "1234"
? PadRight("123456", 8)           // "123456  "
? PadRight("123456", 8, ".")      // "123456.."
? PadRight("123456", 8, 43)       // "123456++"
? PadRight("123"+chr(0)+"456", 8)  // "123"+chr(0)+"456  "

? PAdr("123456", 4)               // "1234"
? PAdr("123456", 8)               // "123456  "
? PAdr("123456", 8, ".")          // "123456.."
? PAdr("123"+chr(0)+"456", 8)     // "123"+chr(0)+"456  "
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:PadLeft(), Padl(), PAdr(), Padc()

# POSALPHA ( )

---

**Syntax:**

**retN = PosAlpha ( expC1, [expL2], [expN3] )**

**Purpose:**

Determines the position of the first alphabetic character in a string

**Arguments:**

<expC1> is the character string that is processed. Embedded zero bytes are supported.

**Options:**

<expL2> designates how the function searches for the first alphabetic character in a string. The default value .F. searches for the first alphabetic character, whilst .T. searches for the first non-alphabetic character.

<expN2> is an optional numeric value specifying the number of characters at the beginning of <expC1> that are excluded from the search. The default value 0 excludes none.

**Returns:**

<retN> is the position of the first alphabetic character in <expC1> when <expL2> is NIL or .F., or the position of the first non- alphabetic character when <expL2> is .T. If such a position is not found, 0 is returned.

**Description:**

Starting from the <expN2> offset in <expC1>, PosAlpha() searches for the first alphabetic (or non-alphabetic) character in <expC1>.

Embedded zero bytes in <expC1> are not considered.

**Example:**

```
cString := "JkLm123"
? PosAlpha(cString)           // 1
? PosAlpha(cString, .T.)      // 5
? PosAlpha(cString, .F., 2)    // 3
? PosAlpha("xxx", .T. )       // 0
? PosAlpha("xxx"+chr(129)+"12", .T. ) // 5
? PosAlpha("xxx"+chr(205)+"12", .T. ) // 4
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:PosLower(), FS2:PosUpper(), FS2:PosRange(), At(), IsAlpha()

# POCHAR ()

---

**Syntax:**

`retC = PosChar ( expC1, expCN2, [expN3] )`

**Purpose:**

Replaces an individual character at a particular position within a string.

**Arguments:**

<expC1> is the character string that is processed.

<expNC2> is either a character or its numeric equivalence which should be replaced in <expC1> at position <expN3>. If the <expCN2> string is longer, only the first character is used.

**Options:**

<expN2> is an optional position at which the <expCN2> character is substituted. The default value is the last position in <expC1>.

**Returns:**

<retC> is the modified string.

**Description:**

PosChar() replaces an individual character within a string. It is very similar to the standard function StrPoke().

**Example:**

```
? PosChar("ABCDEF", "x")           // "ABCDEx"
? PosChar("ABCDEF", "x", 3)         // "ABxDEF"
? PosChar("ABCDEF", 120, 3)         // "ABxDEF"
? StrPoke("ABCDEF", 3, 120)         // "ABxDEF"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

StrPoke(), StrTran(), Substr()

# POSDEL ()

---

**Syntax:**

```
retC = PosDel ( expC1, [expN2], [expN3] )
```

**Purpose:**

Deletes characters at a particular position in a string.

**Arguments:**

<expC1> is the character string that is processed. Embedded zero bytes are supported.

**Options:**

<expN2> is a position from which the deletion begins. If not specified, the deletion begins at the end of the <expC1> string and deletes the specified number of <expN3> characters.

<expN3> is the number of characters to delete, default is 1.

**Returns:**

<retC> is the modified string.

**Description:**

This function permits the removal of <xpN3> of characters from <expC1>, beginning from <expN2>. It avoids the otherwise required concatenation of Left() and Substr() expressions.

**Example:**

```
? PosDel("Parameter", 3, 2)           // "Pameter"
? PosDel("Parameter", , 2)            // "Paramet"

? PosDel("1234567")                   // "123456"
? PosDel("1234567", 7)                 // "123456"
? PosDel("1234567", , 1)              // "123456"

? PosDel("1234567", 3)                 // "124567"
? PosDel("1234567", 3, 2)              // "12567"
? PosDel("1234567", , 2)              // "12345"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:PosIns(), FS2:PosRange(), FS2:RangeRem(), Substr()

# POSDIFF ()

---

**Syntax:**

```
retN = PosDiff ( expC1, expC2, [expN3] )
```

**Purpose:**

Finds the first position from which two strings differ.

**Arguments:**

<expC1> is the string that is processed. Embedded zero bytes are supported.

<expC2> is the string that is compared with <expC1>. Embedded zero bytes are supported.

**Options:**

<expN3> is an optional number of characters at the beginning of both <expC1> and <expC2> that are excluded from the comparison. The default value 0 excludes none.

**Returns:**

<retN> is the position where <expC1> and <expC2> differ. If both strings are equal, or <expN3> is longer than one of the strings, 0 is returned.

**Description:**

PosDiff() compares two strings and determines from which position the first difference occurs. The <expN3> parameter allows you to exclude a particular number of characters from the beginning of both strings from the search. Strings of different lengths can be compared too with each other.

**Example:**

```
cString1 := "12345678"
cString2 := "1234x678"
cString3 := "a234x678"
cString4 := "A234x678"

? PosDiff(cString1, cString2)           // 5
? PosDiff(cString1, cString3)           // 1
? PosDiff(cString1, cString3, 1)         // 5
? PosDiff(cString2, cString3, 1)         // 0
? PosDiff(cString3, cString4)           // 1
? PosDiff(upper(cString3), upper(cString4)) // 0

? PosDiff("AB", "ABC")                  // 3
? PosDiff("ABC", "AB")                   // 3
? PosDiff("ABC", "")                    // 1
? PosDiff("ABC", "AB", 7)                // 0
```

**Classification:** add-on library, programming

**Compatibility:** Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:** FS2:PosEqual()



# POSEQUAL ()

---

**Syntax:**

**retN = PosEqual ( expC1, expC2, [expN3], [expN4] )**

**Purpose:**

Finds the first position at which two strings are the same.

**Arguments:**

<expC1> is the string that is processed. Embedded zero bytes are supported.

<expC2> is the string that is compared with <expC1>. Embedded zero bytes are supported.

**Options:**

<expN3> is an optional number of characters that must be the same. The default value is the length of the shorter string less <expN4>.

<expN4> is an optional number of characters at the beginning of both <expC1> and <expC2> that are excluded from the comparison. The default value 0 excludes none.

**Returns:**

<retN> is the position where <expC1> and <expC2> are equal. If no corresponding characters are found, or <expN4> is longer than one of the strings, 0 is returned.

**Description:**

PosEqual() compares two strings and determines at which position the first difference occurs. With <expN3> you may specify the minimal length of equivalence. The <expN4> parameter allows you to exclude a particular number of characters from the beginning of both strings from the search. Strings of different lengths can be compared too with each other.

**Example:**

```
cString1 := "ABCDEFGHI"
cString2 := "XYZDEKLMN"
? PosEqual(cString1, cString2)           // 0
? PosEqual(cString1, cString2, 2)       // 4
? PosEqual(cString1, cString2, 1, 4)    // 5

? PosEqual("AB", "ABC")                 // 1
? PosEqual("xAB", "yABC", 2)            // 2
? PosEqual("xAB", "yABC", , 1)          // 2
? PosEqual("xABC", "yAB", 2)            // 2
? PosEqual("xABC", "")                  // 0

? PosEqual("yyAByy", "xxABC", 2)        // 3
? PosEqual("xABC", "AAAAA", 1)          // 2
```

**Classification:** add-on library, programming

**Compatibility:** Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:** FS2:PosDiff()

# POSINS ( )

---

**Syntax:**

`retC = PosIns ( expC1, expC2, [expN3] )`

**Purpose:**

Inserts characters at a particular position within a string.

**Arguments:**

<expC1> is the string that is processed. Embedded zero bytes are supported.

<expC2> is the string that is inserted into <expC1>. Embedded zero bytes are supported.

**Options:**

<expN3> is the position where the new <expC2> characters are inserted within <expC1>. The default value inserts the characters in front of the last character of <expC1>.

**Returns:**

<retC> is the modified string, or <retC1> if <expN3> is out of range.

**Description:**

PosIns() inserts characters into an existing string. The <expC2> characters are inserted into the <expC1> at the location specified by <expN3>.

**Example:**

```
? PosIns("abcdefgh", "123")           // "abcdefgh123h"
? PosIns("abcdefgh", "123", 1)        // "123abcdefgh"
? PosIns("abcdefgh", "123", 2)        // "a123bcdefgh"
? PosIns("abcdefgh", "123", 8)        // "abcdefgh123h"
? PosIns("abcdefgh", "123", 9)        // "abcdefgh123"
? PosIns("abcdefgh", "123", 10)       // "abcdefgh"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:PosDel(), PosRepl(), Substr()

# POSLOWER ()

---

**Syntax:**

**retN = PosLower ( expC1, [expL2], [expN3] )**

**Purpose:**

Finds the position of the first lower case alphabetic character.

**Arguments:**

<expC1> is the character string that is processed. Embedded zero bytes are supported.

**Options:**

<expL2> designates how the function searches for the first lower case alphabetic character in <expC1>. The default value .F. searches for the first lower case alphabetic character, whilst .T. searches for the first non-alphabetic or non lower case character.

<expN2> is an optional numeric value specifying the number of characters at the beginning of <expC1> that are excluded from the search. The default value 0 excludes none.

**Returns:**

<retN> is the position of the first lower case alphabetic character in <expC1> when <expL2> is NIL or .F., or the position of the first non-alphabetic or non lowercase character when <expL2> is .T. If such a position is not found, 0 is returned.

**Description:**

Starting from the <expN2> offset in <expC1>, PosLower() searches for the first lower case alphabetic character (or the opposite of) in <expC1>.

**Example:**

```
cString := "123ABCuabc"
? PosLower(cString)           // 7
? PosLower(cString, .T.)      // 1
? PosLower(cString, .F., 7)   // 8
? PosLower(cString, .F., 2)   // 7

? PosLower("AB5" )           // 0
? PosLower("AB5", .T. )      // 1
? PosLower("AB5"+chr(129)+"xy" ) // 4
? PosLower("AB5"+chr(205)+"xy" ) // 5
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:PosAlpha(), FS2:PosUpper(), FS2:PosRange(), At(), IsLower()

# POSRANGE ()

---

## Syntax:

```
retN = PosRange ( expCN1, expCN2, expC3, [expL4],  
                  [expN5] )
```

## Purpose:

Finds the position of the first lower case alphabetic character.

## Arguments:

<expCN1> is the lower boundary character or it numeric equivalence that is searched in <expC3>. Supported are characters chr(0) to chr(255).

<expCN2> is the upper boundary character or it numeric equivalence that is searched in <expC3>. Supported are characters chr(0) to chr(255).

<expC3> is the character string that is processed. Embedded zero bytes are supported.

## Options:

<expL4> designates how the function searches for an character within the range <expCN1> to <expCN2>. The default value .F. (or NIL) searches for the first character within this range, whilst .T. searches for the first character outside of the range.

<expN5> is an optional numeric value specifying the number of characters at the beginning of <expC3> that are excluded from the search. The default value 0 excludes none.

## Returns:

<retN> is the position of the first character within (or outside of) the given range. If such a position is not found, 0 is returned.

## Description:

Starting from the <expN5> offset in <expC3>, PosRange() searches for the first character in the range <expCN1>..<<expCN2> (or the opposite of) within the <expC3> string.

## Example:

```
cString := "abcABC" + chr(0) + "x" + chr(10) + "zzz"  
  
? PosRange("A", "Z", cString)           // 4  
? PosRange("A", "Z", cString, .T.)       // 1  
? PosRange("A", "Z", cString, NIL, 7)    // 0  
? PosRange("A", "Z", cString, .T., 7)    // 8  
? PosRange(chr(0), chr(31), cString)     // 7  
? PosRange(chr(0), chr(31), cString, .T.) // 1  
? PosRange(0, 31, cString, .T., 10)      // 11
```

## Classification:

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CT3 supports only characters for <expNC1> and <expNC2>.

**Related:**

FS2:PosAlpha(), FS2:PosUpper(), FS2:PosLower(), FS2:RangeRem(), At()

# POSREPL ()

---

**Syntax:**

`retC = PosRepl ( expC1, expC2, [expN3] )`

**Purpose:**

Replaces one or more characters from a certain position.

**Arguments:**

<expC1> is the string that is processed and partially replaced by <expC2>. Embedded zero bytes are supported.

<expC2> is the replacement string. Embedded zero bytes are supported.

**Options:**

<expN3> is the starting position in <expC1> from which <expC2> replaces the original string. If not specified, the default setting overwrites the end of the <expC1> string, i.e. is calculated as  $\max(1, \text{len}(\text{expC1}) - \text{len}(\text{expC2}) + 1)$

**Returns:**

<retC> is the modified string. On failure, the original <expC1> is returned.

**Description:**

With PosRepl(), you can replace a range of characters within <expC1> with another string <expC2>. The replacement begin at <expN3>. It can in some cases extend the size of the original string or overwrite it in fully, see examples below.

**Example:**

```
? PosRepl("123456", "xxx")           // "123xxx"
? PosRepl("123456", "xxx", 3)         // "12xxx6"
? PosRepl("123456", "xxx", 5)         // "1234xxx"
? PosRepl("123456", "xxx", 6)         // "12345xxx"
? PosRepl("123456", "xxx", 7)         // "123456xxx"
? PosRepl("123456", "xxx", 8)         // "123456"
? PosRepl("12", "xxx")                // "xxx"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:PosIns(), FS2:PosDel(), Strtran()

# POSUPPER ()

---

**Syntax:**

**retN = PosUpper ( expC1, [expL2], [expN3] )**

**Purpose:**

Finds the position of the first upper case alphabetic character.

**Arguments:**

<expC1> is the character string that is processed. Embedded zero bytes are supported.

**Options:**

<expL2> designates how the function searches for the first upper case alphabetic character in <expC1>. The default value .F. searches for the first upper case alphabetic character, whilst .T. searches for the first non-alphabetic or non upper case character.

<expN2> is an optional numeric value specifying the number of characters at the beginning of <expC1> that are excluded from the search. The default value 0 excludes none.

**Returns:**

<retN> is the position of the first upper case alphabetic character in <expC1> when <expL2> is NIL or .F., or the position of the first non-alphabetic or non upper case character when <expL2> is .T. If such a position is not found, 0 is returned.

**Description:**

Starting from the <expN2> offset in <expC1>, PosUpper() searches for the first upper case alphabetic character (or the opposite of) in <expC1>.

**Example:**

```
cString := "123abcUABC"
? PosUpper(cString)           // 7
? PosUpper(cString, .T.)      // 1
? PosUpper(cString, .F., 7)   // 8

? PosUpper("ab5" )           // 0
? PosUpper("ab5", .T. )      // 1
? PosUpper("ab5"+chr(142)+"XY" ) // 4
? PosUpper("ab5"+chr(205)+"XY" ) // 5
? PosUpper("ab5"+chr(0)+"XY" ) // 5
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:PosAlpha(), FS2:PosLower(), FS2:PosRange(), At(), IsUpper()

# RANGEREM ()

---

## Syntax:

```
retC = RangeRem ( expCN1, expCN2, expC3, [expL4],  
                 [expN5] )
```

## Purpose:

Deletes characters that are within a specified ASCII code range.

## Arguments:

<expCN1> is the lower boundary character or its numeric equivalence that is searched and deleted in <expC3>. Supported are characters chr(0) to chr(255) or values 0 to 255. "" is equivalent to 0.

<expCN2> is the upper boundary character or its numeric equivalence that is searched and deleted in <expC3>. Supported are characters chr(0) to chr(255) or values 0 to 255. "" is equivalent to 0. If <expCN2> has a lower value than <expCN1>, the range is reverse and similar as with set <expL4> = .T.

<expC3> is the character string that is processed. Embedded zero bytes are supported.

## Options:

<expL4> designates how the function searches for a character within the range <expCN1> to <expCN2>. The default value .F. (or NIL) removes all characters within this range, whilst .T. removes all characters that does not fit to the range, i.e. leaves only characters that are within the specified range.

<expN5> is an optional numeric value specifying the number of characters at the beginning of <expC3> that are skipped and hence excluded from the search and deletion. The default value 0 excludes none.

## Returns:

<retC> is the modified string. If the corresponding character's boundary is not found, the original <expC3> string is returned.

## Description:

Starting from the <expN5> offset in <expC3>, RangeRem() searches for all character in the range <expCN1>..<<expCN2> (or the opposite of) in the <expC3> string and if such are found, deletes them.

## Example:

```
cString := "abcABC" + chr(0) + "12zzz"      // "abcABC?12zzz"  
? RangeRem( 0, 31, cString)                  // "abcABC12zzz"  
? RangeRem(chr(0), chr(31), cString)         // "abcABC12zzz"  
? RangeRem("0", "9", cString)                // "abcABC?zzz"  
? RangeRem("A", "Z", cString)                // "abc?12zzz"  
? RangeRem( 0, 64, cString)                  // "abcABCzzz"  
? RangeRem("A", "Z", cString, .T.)           // "ABC"  
? RangeRem( 91, 64, cString)                 // "ABC"  
? RangeRem( "", "Z", cString, , 4)           // "abcA12zzz"
```



```
? RangeRem("A", "Z", cString, .T., 4)      // "abcABC"  
? RangeRem( 10, 11, cString)              // "abcABC?12zzz"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CT3 supports only the first three arguments.

**Related:**

FS2:RangeRepl(), FS2:PosRange(), FS2:FS2:PosDel(), FS2:PosRepl(), Substr()

# RANGEREPL ()

---

## **Syntax:**

```
retC = RangeRepl ( expCN1, expCN2, expC3, expCN4,  
                  [expL5], [expN6] )
```

## **Purpose:**

Replaces characters within a specified ASCII code range with a particular character

## **Arguments:**

<expCN1> is the lower boundary character or its numeric equivalence that is searched and replaced in <expC3> by <expCN4>. Supported are characters chr(0) to chr(255) or values 0 to 255. "" is equivalent to 0.

<expCN2> is the upper boundary character or its numeric equivalence that is searched and replaced in <expC3> by <expCN4>. Supported are characters chr(0) to chr(255) or values 0 to 255. "" is equivalent to 0. If <expCN2> has a lower value than <expCN1>, the range is reverse and similar as with set <expL5> = .T.

<expC3> is the character string that is processed. Embedded zero bytes are supported.

<expCN4> is the character or its numeric equivalence that replaces all characters in <expC3> that are (or are not) in the specified range. Supported are characters chr(0) to chr(255) or values 0 to 255.

## **Options:**

<expL5> designates how the function searches for a character within the range <expCN1> to <expCN2>. The default value .F. (or NIL) replaces all characters within this range by <expCN4>, whilst .T. replaces all characters that does not fit to the range.

<expN6> is an optional numeric value specifying the number of characters at the beginning of <expC3> that are skipped and hence excluded from the search and replacement. The default value 0 excludes none.

## **Returns:**

<retC> is the modified string. If the corresponding character's boundary is not found, the original <expC3> string is returned.

## **Description:**

Starting from the <expN5> offset in <expC3>, RangeRepl() searches for all character in the range <expCN1>..<<expCN2> (or the opposite of) in the <expC3> string and if such are found, replaces them by the <expCN4> character.

**Example:**

```
cString := "a" + chr(5) + "b" + chr(9)
? RangeRepl(chr(0), chr(31), cString, ".") // "a.b."
? RangeRepl("", 64, "123400", "0") // "000000"
? RangeRepl("0", "7", "089 - 78 67 43", "9") // "989 - 98 99 99"

? cString := "A()&BC/+D" // "A()&BC/+D"
? RangeRepl("A", "Z", cString, ".") // ".()&../+."
? RangeRepl("Z", "A", cString, ".") // "A...BC..D"
? RangeRepl("A", "Z", cString, ".", .T.) // "A...BC..D"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CT3 supports only the first four arguments.

**Related:**

FS2:RangeRem(), FS2:PosRange(), FS2:FS2:PosDel(), FS2:PosRepl(), Substr()

# REMA11 ( )

---

## Syntax:

**retC** = **RemAll** ( **expC1**, [**expCN2**], [**expL3**] )

## Purpose:

Removes particular characters from the beginning and end of a string or from the entire string.

## Arguments:

<**expC1**> is the string that is processed. Embedded zero bytes are supported.

## Options:

<**expCN2**> is the character or its numeric equivalence that is removed from the beginning and end of <expC3>, or from the entire string. If not specified, the default is space = chr(32). Null string "" is equivalent to 0.

<**expL3**> is an optional logical value. If .F. or NIL (the default), RemAll() removes the leading and trailing characters only. If .T., the function removes all <expCN2> from the <expC1> string and acts similarly to Strtran(expC1, expCN2, "") but accepts also binary 0.

## Returns:

<**retC**> is the modified string.

## Description:

RemAll() is equivalent to the standard function Alltrim() when <expCN2> and <expL3> are omitted. But as opposite to Alltrim(), RemAll() can remove any character you choose.

## Example:

```
? RemAll (" 1 2 3 ") // "1 2 3"
? Alltrim(" 1 2 3 ") // "1 2 3"
? RemAll ("001020300", "0") // "10203"
? RemAll ("001020300", "0", .T.) // "123"
? Strtran("001020300", "0", "") // "123"

cString := chr(0) + "abc" + chr(0) + "def" + chr(0)
? cString // "?abc?def?"
? RemAll(@cString, chr(0)) // "abc?def"
? RemAll(@cString, chr(0), .T.) // "abcdef"
```

## Classification:

add-on library, programming

## Compatibility:

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only the first two arguments.

## Related:

FS2:RemLeft(), FS2:RemRight(), FS2:ReplAll(), FS2:PosDel(), Alltrim(), Strtran(), Substr()

# REMLEFT ()

---

**Syntax:**

`retC = RemLeft ( expC1, [expCN2] )`

**Purpose:**

Removes particular characters from the beginning of a string.

**Arguments:**

<expC1> is the string that is processed. Embedded zero bytes are supported.

**Options:**

<expCN2> is the character or its numeric equivalence that is removed from the beginning of <expC3>. If not specified, the default is space = chr(32). "" is equivalent to 0.

**Returns:**

<retC> is the modified string.

**Description:**

RemLeft() is equivalent to the standard function Ltrim() when <expCN2> is omitted. But as opposite to Ltrim(), RemLeft() can remove any character you choose.

**Example:**

```
? RemLeft(" 1 2 3 ") // "1 2 3"
? Ltrim (" 1 2 3 ") // "1 2 3"
? RemLeft("001020300", "0") // "1020300"
? RemLeft(chr(0) + "abc", "") // "abc"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:RemAll(), FS2:RemRight(), FS2:ReplLeft(), FS2:PosDel(), Ltrim()

# REMRIGHT ()

---

**Syntax:**

`retC = RemRight ( expC1, [expCN2] )`

**Purpose:**

Removes particular characters at the end of a string.

**Arguments:**

<expC1> is the string that is processed. Embedded zero bytes are supported.

**Options:**

<expCN2> is the character or its numeric equivalence that is removed at the end of <expC3>. If not specified, the default is space = chr(32). "" is equivalent to 0.

**Returns:**

<retC> is the modified string.

**Description:**

RemRight() is equivalent to the standard function Rtrim() when <expCN2> is omitted. But as opposite to Rtrim(), RemRight() can remove any character you choose.

**Example:**

```
? RemRight(" 1 2 3 ") // " 1 2 3"
? Rtrim (" 1 2 3 ") // " 1 2 3"
? RemRight("001020300", "0") // "0010203"
? RemRight("abc" + chr(0), 0) // "abc"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:RemAll(), FS2:RemLeft(), FS2:ReplRight(), Rtrim()

# REPLALL ()

---

## Syntax:

```
retC = ReplAll ( expC1, expCN2, [expCN3], [expL4] )
```

## Purpose:

Exchanges particular characters at the beginning and end of a string, or within the entire string.

## Arguments:

<expC1> is the string that is processed. Embedded zero bytes are supported.

<expCN2> is the character or its numeric equivalence that replaces all spaces or <expCN3> characters at the begin and end of <expC1> or within the whole string. Null string "" is interpreted as space " ".

## Options:

<expCN3> is the character or its numeric equivalence that is searched and replaced in <expC1> by <expCN2>. If not specified, the default is " " space = chr(32). Null string "" is equivalent to chr(0).

<expL4> is an optional logical value. If .F. or NIL (the default), ReplAll() replaces the leading and trailing characters only. If .T., the function replaces all <expCN3> characters by <expCN2> and act similarly to Strtran(expC1, expCN3, expCN2) but accept also binary 0.

## Returns:

<retC> is the modified string.

## Description:

ReplAll() can be used to exchange all leading and trailing spaces (or character specified by <expCN3>) within the <expC1> string by any other character specified in <expCN2>. It also can replace all <expCN3> characters by <expCN2> in the entire <expC1> string.

## Example:

```
? ReplAll("  abc  ", "-")           // "---abc---"
? ReplAll(" 1 2 3 ", ".")           // "..1 2 3.."
? ReplAll(" 1 2 3 ", ".", .T.)       // "...1.2.3..."
? ReplAll("001020345", " ", "0")     // " 1020345"
? ReplAll("001020300", " ", "0")     // "..10203.."
? ReplAll("001020300", ".", "0", .T.) // "...1.2.3..."
? Strtran("001020300", "0", ".")     // "...1.2.3..."

cString := chr(0) + "abc" + chr(0) + "def" + chr(0)
? cString                               // "?abc?def?"
? ReplAll(@cString, "-", 0)             // "-abc?def-"
? ReplAll(@cString, "-", 0, .T.)        // "-abc-def-"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only the first three arguments.

**Related:**

FS2:ReplLeft(), FS2:ReplRight(), FS2:RangeRepl(), FS2:RemAll(), FS2:PosRepl(), Strtran()



# REPLLEFT ()

---

**Syntax:**

```
retC = ReplLeft ( expC1, expCN2, [expCN3] )
```

**Purpose:**

Exchanges particular characters at the beginning of a string.

**Arguments:**

<expC1> is the string that is processed. Embedded zero bytes are supported.

<expCN2> is the character or it numeric equivalence that replaces all spaces or <expCN3> characters at the begin of <expC1>. Null string "" is interpreted as space " ".

**Options:**

<expCN3> is the character or it numeric equivalence that is searched and replaced in <expC1> by <expCN2>. If not specified, the default is " " space = chr(32). Null string "" is equivalent to chr(0).

**Returns:**

<retC> is the modified string.

**Description:**

ReplLeft() can be used to exchange all leading spaces (or character specified by <expCN3>) within the <expC1> string by any other character specified in <expCN2>.

**Example:**

```
? ReplLeft("  abc  ", "-") // "---abc  "
? ReplLeft(" 1 2 3  ", ".") // "..1 2 3  "
? ReplLeft("001020300", ".", "0") // "..1020300"
? ReplLeft("001020345", " ", "0") // " 1020345"
? ReplLeft(chr(0)+"xyz"+chr(0), "-", "") // "-xyz?"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:ReplAll(), FS2:ReplRight(), FS2:RangeRepl(), FS2:RemAll(), FS2:RemLeft(), FS2:PosRepl(), Substr(), Strtran()

# REPLRIGHT ()

---

**Syntax:**

`retC = ReplRight ( expC1, expCN2, [expCN3] )`

**Purpose:**

Exchanges particular characters at the end of a string.

**Arguments:**

<expC1> is the string that is processed. Embedded zero bytes are supported.

<expCN2> is the character or its numeric equivalence that replaces all spaces or <expCN3> characters at the end of <expC1>. Null string "" is interpreted as space " ".

**Options:**

<expCN3> is the character or its numeric equivalence that is searched and replaced in <expC1> by <expCN2>. If not specified, the default is " " space = chr(32). Null string "" is equivalent to chr(0).

**Returns:**

<retC> is the modified string.

**Description:**

ReplRight() can be used to exchange all trailing spaces (or character specified by <expCN3>) within the <expC1> string by any other character specified in <expCN2>.

**Example:**

```
? ReplRight("  abc  ", "-")           // "  abc---"
? ReplRight(" 1 2 3 ", ".")           // " 1 2 3.."
? ReplRight("001020300", ".", "0")     // "0010203.."
? ReplRight("xyz"+chr(0)+chr(0), "-", "") // "xyz--"
? ReplRight("xyz"+chr(0)+chr(0), "", "") // "xyz "
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:ReplAll(), FS2:ReplLeft(), FS2:RangeRepl(), FS2:RemAll(), FS2:RemRight(), FS2:PosRepl(), Substr(), Strtran()

# RESTTOKEN ()

---

**Syntax:**

```
[retC = ] RestToken ( expC1 )
```

**Purpose:**

Recreates an incremental tokenizer environment.

**Arguments:**

<expC1> is the string that is processed and previously returned by the SaveToken() function.

**Returns:**

<retC> is always "".

**Description:**

RestToken() is belonging to the group of versatile incremental tokenizer functions, initialized by TokenInit() and processed by TokenNext(). RestToken() restores the internal environment and does the opposite of the SaveToken() function. The <expC1> string must originate from the current program run; for example, it cannot be restored from database or .mem file.

**Example:**

```
Here is a small example of incremental tokenizer. The text is
broken
into individual lines, and each line is broken into words:

cText := "Hello world!" + chr(10) + "and the rest of universe"

TokenInit(@cText, chr(10)+chr(13) ) // token separator is CR or LF
WHILE .NOT. TokenEnd()
    cLine := TokenNext(cText)
    ? "Line = '" + cLine + "'"      // display all lines
    Displaywords(cLine)
ENDDO
wait "Done ..."
```

```
FUNCTION Displaywords(cLine)
    local cword
    local cOldEnv := SaveToken()    // save current token = line
    TokenInit(@cLine, " .,:;!?" ) // uses other token separators
    WHILE .NOT. TokenEnd()
        cword := TokenNext(cLine)
        ? "word = '" + cword + "'" // display all single words
    ENDDO
    RestToken(cOldEnv)              // restore current token
RETURN NIL
```

**Classification:** add-on library, programming

**Compatibility:** Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:** FS2:SaveToken(), TokenInit(), TokenNext(), TokenArray()

# SAVETOKEN ()

---

**Syntax:**

`retC = SaveToken ( )`

**Purpose:**

Saves the incremental tokenizer environment to a variable.

**Returns:**

<retC> is a string containing the internal environment for the incremental tokenizer.

**Description:**

SaveToken() is belonging to the group of versatile incremental tokenizer functions, initialized by TokenInit() and processed by TokenNext().

SaveToken() and RestToken() are used for saving and restoring of internal tokenizer environment, e.g. when the current token needs to be further splitted to sub-tokens.

The returned <retC> value is only valid for the currently running program and is meant for use by subsequent RestToken() because it is concerned with internal pointers.

**Example:**

see example in SaveToken()

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:RestToken()

# SETATLIKE ()

---

## Syntax:

**retN = SetAtlike ( [expN1], [expC2] )**

## Purpose:

Provides an additional search mode for all FS2 AT\*() functions and StrDiff().

## Options:

<expN1> is numeric value specifying the ON (1) and OFF (0) setting of the wildcard search. The default value is 0.

<expC2> is an optional wildcard character. The default value is "?" You may use any other character (e.g. "#") when the question-mark is already available in the string as regular character.

## Returns:

<retN> is the current mode at the time of entering this function.

## Description:

All AT\*() functions like AtNum(), AfterAtnum() and so on, as well as the function StrDiff(), operate to find an exact match during the execution of the search sequence. When you set SetAtlike(1), an additional search using wildcard characters is permitted. For every position containing a wildcard character within the search expression, any character can occur in the character string that is searched. The first character of the search expression cannot be a wildcard, and the entire expression cannot consist of wildcards. These restrictions simultaneously avoid unwanted recursions with AtRepl().

The customary "?" has been used as the default wildcard character. However, it can be replaced by using the optional <expC2> parameter. Any character you choose can be set as a wildcard character, increasing the flexibility of this group of functions.

## Example:

```
? SetAtlike(1)           // uses default "?" wildcard character

// Determine last position of the search expression:
? AtNum("ABC", "ABCDEABC123AXCK")           // 6
? AtNum("A?C", "ABCDEABC123AXCK")           // 12

// How often occurs the search expression in the text sequence?
? NumAt("ABC", "ABCDEABC123AXCK")           // 2
? NumAt("A?C", "ABCDEABC123AXCK")           // 3

// Get the text sequence behind the last search occurrence:
? AfterAtnum("ABC", "ABCDEABC123AXCK")       // 123AXCK
? AfterAtnum("A?C", "ABCDEABC123AXCK")       // K

// Get the text sequence before the last search occurrence:
? BeforAtnum("ABC", "ABCDEABC123AXCK")       // ABCDE
? BeforAtnum("A?C", "ABCDEABC123AXCK")       // ABCDEABC123
```

```

// Adjust text to 15 columns, split at the last search occurrence
? AtAdjust("ABC", "ABCDEABC123AXCK", 15)    // ABCDE   ABC123AXCK
? AtAdjust("A?C", "ABCDEABC123AXCK", 15)    // ABCDEABC123   AXCK

// Searches for a sequence within a string and replaces it
CsetAtMupa(.T.)                               // multiple replacement
? AtRepl("D?", "ABCDEABC123AXDK", "Dx")     // ABCDXABC123AXDX

// wildcard may reduce the valence in StrDiff()
? StrDiff("ABC", "AXC")                       // 1
? StrDiff("??C", "AXC")                       // 0
? StrDiff("axC", "AX")                        // 3 (2 replace, 1 delete)
? StrDiff("??C", "AX")                        // 1 (delete)
? StrDiff("axC", "AX", 3,6,1)                 // 12 (2 replace, 1 delete)
? StrDiff("??C", "AX", 3,6,1)                 // 6 (1 delete)

```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:AtAdjust(), FS2:AtNum(), FS2:AtAfternum(), FS2:AtRepl(), FS2:BeforAtnum(),  
FS2:NumAt(), FS2:StrDiff()

# STRDIFF ()

---

## Syntax:

```
retN = StrDiff ( expC1, expC2, [expN2], [expN3],  
                [expN4] )
```

## Purpose:

Finds similarity between two strings (Levenshtein Distance).

## Arguments:

<expC1> is the string that is compared with <expC2>. Embedded zero bytes are supported.

<expC2> is the string that is compared with <expC1>. Embedded zero bytes are supported.

## Options:

<expN3> is an optional numeric value specifying the cost of replacement. The default value is 1 (and is 3 in CA3 tools).

<expN4> is an optional numeric value specifying the cost of removal. The default value is 1 (and is 6 in CA3 tools).

<expN5> is an optional numeric value specifying the cost of insert. The default value is 1 (same as in CA3 tools).

## Returns:

<retN> is a numeric value corresponding to the difference between the <expC1> and <expC2>. If <expN3>..<>expN5> are 1 (the default), <retN> represents the number of insert, replace and delete operations needed to transform <expC1> into <expC2>, otherwise the result is classified "cost" of such required operation. Zero value signals an equivalence of both strings.

## Description:

This string comparison use the Levenshtein Distance Algorithm, which reports the number of insert, replace and delete operations (e.g. keystrokes) needed to transform <expC1> into <expC2>. You may additionally weight (classify) the required operation cost (valence) by a corresponding factor in <expN3> to <expN5> parameters.

Note: this algorithm is complex and memory hungry (it requires at least 6-times the size of <expC1> and <expC2>) and the amount of calculation steps grows quadratic. Although FS2 does not limit the string sizes (supported are up to 2 Gigabytes each), it is highly recommended to limit the size by using Substr() or Left() instead of comparing very large strings by StrDiff(), as well as to pass large strings by reference; see example and comparison below.

## Example:

```
? StrDiff("ABC", "ABC")           // 0 differences  
? StrDiff("ABC", "ADC")           // 1 char differs  
? StrDiff("ABC", "AEC")           // 1 char differs
```

```

? StrDiff("CBA", "ABC")           // 2 chars differs
? StrDiff("ABC", "AXBC")          // 1 char differs
? StrDiff("AXBC", "ABC")          // 1 char differs

? StrDiff("ABC", "ABC", 3,6,1)    // 0 = both are equivalent
? StrDiff("ABC", "ADC", 3,6,1)    // 3 = Replace 1 character
? StrDiff("ABC", "AEC", 3,6,1)    // 3 = Replace 1 character
? StrDiff("CBA", "ABC", 3,6,1)    // 6 = Replace 2 characters
? StrDiff("ABC", "AXBC", 3,6,1)   // 1 = Insert 1 character
? StrDiff("AXBC", "ABC", 3,6,1)   // 6 = Delete 1 character

cStr1 := space(100) + "x" + space(1000000) // size 1mb x 1mb =
cStr2 := space(100) + "y" + space(1000000) // 10^12 calculations
? StrDiff(@cStr1, @cStr2)           // 1 (some minutes)
? StrDiff(left(@cStr1,1000), left(@cStr2,1000)) // 1 (some
1/100sec)
? StrDiff(left(@cStr1,110), left(@cStr2,110)) // 1 (some
Millisec)
? PosDiff(@cStr1, @cStr2)           // 101 (some
Millisec)
? cStr1 == cStr2                   // .F. (some
Microsec)

```

### **Classification:**

add-on library, programming

### **Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools use weighted default settings for <expN3> and <expN4> and limits the strings to max 181 bytes each.

### **Related:**

FS2:PosDiff(), FS2:PosEqual(), FS2:SetAlike(), logical string comparison by == and <, >, <=, >=, != operators



# STRSWAP ()

---

## Syntax:

```
retN = StrSwap ( @expC1, @expC2 )
retN = StrSwap ( @expC1, expC2 )
retN = StrSwap ( expC1, @expC2 )
```

## Purpose:

Interchanges two strings.

## Arguments:

<expC1> is the string that is interchanged by <expC2>. Embedded zero bytes are supported.

<expC2> is the string that is interchanged by <expC1>. Embedded zero bytes are supported.

## Returns:

<retN> is 1 if only the first string could be exchanged, 2 if only the second string could be exchanged, 3 if both are swapped, and 0 when both strings remain unchanged.

## Description:

StrSwap() interchanges the string content of <expC1> and <expC2> when both parameters are passed by reference. If only one parameter is passed by reference, only this string is changed. When one of the strings is a constant or database field, its content is copied to the other string, if possible. If both parameters are passed by value (i.e. without the @ prefix), are constants or fields, nothing is done.

If you wish to swap other than character variables, use assignment vTmp := var1 ; var1 := var2 ; var2 := vTmp which can also be used for strings, but is here not so effective as the StrSwap() function.

## Example:

```
cStr1 := "123456"
cStr2 := "ABCDEFGHIJKLM"
? StrSwap(cStr1, cStr2)           // 0
? cStr1, cStr2                   // "123456" "ABCDEFGHIJKLM"

? StrSwap(@cStr1, @cStr2)        // 3
? cStr1, cStr2                   // "ABCDEFGHIJKLM" "123456"

cStr1 := "123456"
cStr2 := "ABCDEFGHIJKLM"
? StrSwap(cStr1, @cStr2)         // 2
? cStr1, cStr2                   // "123456" "123456"

cStr2 := "ABCDEFGHIJKLM"
? StrSwap("123456", @cStr2)      // 2
? cStr2                           // "123456"

? StrSwap(@cStr1, field->name)   // 1
```

```
? cStr1 // "John Miller"
? field->name // "John Miller"

? StrSwap("1234", "ABCDEFGH") // 0
? StrSwap("1234", field->name) // 0
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CT3 returns only fix "". Note that FS2 do what it should <g>, i.e. it really interchanges the string content and handles also constants & fields. CT3 often produce garbage, even documented in the CT3 manual.

**Related:**

LNG.2.6.4, string assignment by using the := or = operator

# STRTOHEX()

---

**Syntax:**

```
retC = StrToHex ( expC1 )
```

**Purpose:**

Converts a byte sequence into hexadecimal form.

**Arguments:**

<expC1> is a string whose bytes are converted to hex representation. Embedded zero bytes are supported.

**Returns:**

<retC> is the hex representation of <expC1> or "" on error.

**Description:**

StrToHex() returns a character string that contains each byte of the <expC1> string in its two-character, hexadecimal form. The returned string is exactly twice as long as <expC1> or is empty on error.

The reverse function is HexToStr(). For converting numeric value into hex string, use the standard FlagShip function Num2hex().

**Example:**

```
? StrToHex( chr(27) + "A1a" + chr(10) ) // "1B4131610A"  
? "0x" + StrToHex( space(2) )           // "0x2020"  
? StrToHex( "" )                        // ""
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:Hex2Str(), Hex2Num(), Num2Hex()

# TABEXPAND ()

---

**Syntax:**

```
retC = TabExpand ( expC1, [expN2], [expCN3] )
```

**Purpose:**

Converts tabs to spaces.

**Arguments:**

<expC1> is the string that is processed. Embedded zero bytes are supported.

**Options:**

<expN2> is the tab width. The default value is 8. If <expN2> is a character, and only two parameters are passed, this parameter is then interpreted as <expCN3>.

<expCN3> is the character or its numeric equivalence that is used to expand TABs. If not specified, the default is " " space = chr(32). Null string "" is interpreted as space.

**Returns:**

<retC> is the modified string.

**Description:**

TabExpand() replaces all TAB characters (^I = CHR(9) = 0x09) in the <expC1> string with one or more spaces or <expCN3> characters to fit the next tab stop <expN2> width.

Carriage return = chr(13) and line feed = chr(10) as well as the soft-carriage return = chr(128+13) and soft-line feed = chr(128+10) are considered as new line and reset the current column count to 0. SetTabs() does not affect and is not affected by this function.

If you wish not to expand but to replace each TAB with single space, use Strtran(expC1, chr(9), " ") instead.

**Example:**

```
#define TAB chr(9)
#define LF chr(10)
? "01234567!9012345!7890123!5678901!"

? TabExpand("a"+TAB+"!" +TAB+TAB+"!")
? TabExpand("ab"+TAB+"!" +TAB+"!++++++!" +TAB+"!")
? TabExpand("abcdef"+TAB+"!" +TAB+"!gh"+TAB+"!")
? TabExpand("abcd"+TAB+"!x(LF)" + LF + "x"+TAB+"!" +TAB+"!")

? TabExpand("1234567"+TAB+"123"+TAB+"123456789"+TAB+"x", 8, ".")
? TabExpand(replicate(TAB+"x",8), 4)
? TabExpand("aa"+TAB+"abc"+TAB+"abcd"+TAB+"abc"+TAB+"abcdefghij"+
            TAB+"a", 4)
```

```
// will display:

01234567!9012345!7890123!5678901!
a          !                      !
ab         !          !+++++++!          !
abcdef    !          !gh        !
abcd      !x(LF)
x          !
1234567.123.....123456789.....x
      x  x  x  x  x  x  x  x  x
aa  abc  abcd  abc  abcdefghij  a
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:TabPack()

# TABPACK ()

---

## Syntax:

```
retC = TabPack ( expC1, [expN2], [expCN3] )
```

## Purpose:

Converts spaces in tabs.

## Arguments:

<expC1> is the string that is processed. Embedded zero bytes are supported.

## Options:

<expN2> is the tab width. The default value is 8. If <expN2> is a character, and only two parameters are passed, this parameter is then interpreted as <expCN3>.

<expCN3> is the character or its numeric equivalence that is replaced by TABs. If not specified, the default is " " space = chr(32). Null string "" is interpreted as space.

## Returns:

<retC> is the modified string.

## Description:

This function does not simply exchange a simple sequence of the same characters for a TAB; instead, it takes into account the true tab positions. If a space or the <expCN3> character is found at a TAB position, and the same character precedes it, TabPack() replaces this sequence with a TAB = chr(9).

Carriage return = chr(13) and line feed = chr(10) as well as the soft-carriage return = chr(128+13) and soft-line feed = chr(128+10) are considered as new line and reset the current column count to 0. SetTabs() does not affect and is not affected by this function.

If another TABs already exists, you may expand them first by TabExpand() with the old TAB width, and insert then new TABs by TabPack() with another width.

## Example:

```
? TabPack("AAAAAA*", "*")      + "<"      // AAAAAA*<
? TabPack("AAAAA***", "*")      + "<"      // AAAAA  <
? TabPack("AAAAA*****", "*")    + "<"      // AAAAA  **<
cText := "ABCD+" + CRLF + "++---+++++"    // ABCD+
? TabPack(cText, 4, "+")          + "<"      // ++---  ++<

cStr1 := "--"+TAB+"!" + TAB + TAB + "!"
cStr2 := TabExpand(cStr1)
cStr3 := TabPack(cStr2)
? cStr3 == cStr1                // .T.
? TabExpand(cStr3) == cStr2      // .T.

? cStr1, len(cStr1)              // "--      !           !"    7
? cStr2, len(cStr2)              // "--      !           !"    25
? cStr3, len(cStr3)              // "--      !           !"    7
? strtran(cStr1, TAB, "T")       // "--T!TT!"
? strtran(cStr2, TAB, "T")       // "--      !           !"    7
```

```
? strtran(cStr3, TAB, "T") // "--T!TT!"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:TabExpand()

# TOKEN ()

---

**Syntax:**

```
retC = Token ( expC1, [expC2], [expN3], [expN4],  
              [expN5] )
```

**Purpose:**

Selects the n-th token from a string.

**Arguments:**

<expC1> is the string that is processed. Embedded zero bytes are supported.

**Options:**

<expC2> is a list of delimiters used to identify the tokens. If not specified, the default <expC2> delimiters are:

```
" ,. ; : ! ? \ / < > ( ) # ^ % & + - * " + chr(9) + chr(13) + chr(10) +  
chr(26) + chr(138) + chr(141) + chr(4) + chr(0)
```

Embedded zero bytes are supported.

<expN3> is optional occurrence counter, specifying which token position in <expC1> is determined. The default value is the last occurrence.

<expN4> is a max number of delimiters (or delimiter groups) before empty token is returned. The default is 0 indicating that empty tokens are not taken into account. A value of 1 will create even empty token for each delimiter (or delimiter group) found. Value of 2 will create empty token when at least two subsequent delimiters (or groups) are detected, and so forth.

<expN5> is a number of subsequent delimiter characters in <expC2> that count as one delimiter group. The default value is 1, i.e. each character in <expC2> counts as separate delimiter.

Note: If the <expC2> is numeric, a parameter shift is assumed and <expC2> is treated as <expN3>, <expN3> as <expN4> and <expN4> as <expN5>. Such a parameter shift is a bad programming technique and not suggested, but handled by FS2 for compatibility purposes to CT3.

**Returns:**

<retC> is the token found at position <expN3> or at the last position in <expC1>. If nothing is found, "" is returned.

**Description:**

A token is a sequence of characters between two (same or different) delimiters (or delimiter group), or between delimiter and begin/end of the string. The Token() function allows you to break down date and time strings, sentences, file names and paths, etc.

Token() searches the <expC1> string for the standard or specified delimiter. If <expN5> is greater than 1, the subsequent character(s) in <expC1> is checked for



match as delimiter group. When <expN4> is specified, it looks for subsequent <n> matching delimiters from <expC2>. If a delimiter or a delimiter group is found, a substring containing the last or <expN3>-th token is determined and returned.

This Token() function is not a part of the group of the versatile incremental tokenizer functions, initialized by TokenInit() and processed by TokenNext(), but is designed for a stand-alone search for tokens.

**Example:**

```
cStr := "This is FlagShip's FS2-Toolbox . "  
? Token(cStr)                // "Toolbox"  
? Token(cStr, , 3)           // "FlagShip's"  
? Token(cStr, , 4)           // "FS2"  
? Token(cStr, , 9)           // ""  
  
cStr := "one,two,,four"  
for ii := 1 to NumToken(cStr)  
    ? Token(cStr, , ii)       // "one"  "two"  "four"  
next  
for ii := 1 to NumToken(cStr,"", 1)  
    ? Token(cStr, "", ii, 1)  // "one"  "two"  ""   "four"  
next  
  
cStr := "one, two,,four,...,six,,eight"  
? NumToken(cStr)             // 5  
? NumToken(cStr, , 1)        // 13  
? Token(cStr, , 4)           // "six"  
? Token(cStr, ",.", 9, 1)    // "six"  
  
? cStr := CharRem(".", cStr)  // "one, two,,four,,six,,eight"  
? NumToken(cStr)             // 5  
? NumToken(cStr, , 1)        // 9  
? Token(cStr, , 4)           // "six"  
? Token(cStr, "", 6, 1)      // "six"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only the first four arguments.

**Related:**

FS2:TokenArray(), FS2:AtToken(), FS2:NumToken(), FS2:TokenSep(),  
FS2:TokenUpper(), FS2:TokenLower(), FS2:TokenSep()

# TOKENARRAY ()

---

## Syntax:

```
retA = TokenArray (expC1, [expC2], [expN3],  
                  [expN4] )
```

## Purpose:

Create an array of all tokens in a string.

## Arguments:

<expC1> is the string that is processed. Embedded zero bytes are supported.

## Options:

<expC2> is a list of delimiters used to identify the tokens. If NIL, not specified or "", the default <expC2> delimiters are:

```
" ,. ; : ! ? \ / < > ( ) # ^ % & + - * " + chr(9) + chr(13) + chr(10) +  
chr(26) + chr(138) + chr(141) + chr(4) + chr(0)
```

Embedded zero bytes are supported.

<expN3> is a max number of delimiters (or delimiter groups) before empty token is returned. The default is 0 indicating that empty tokens are not taken into account. A value of 1 will create even empty token for each delimiter (or delimiter group) found. Value of 2 will create empty token when at least two subsequent delimiters (or groups) are detected, and so forth.

<expN4> is a number of subsequent delimiter characters in <expC2> that count as one delimiter group. The default value is 1, i.e. each character in <expC2> counts as separate delimiter.

## Returns:

<retCA> is one-dimensional array containing all tokens of <expC1>. If nothing is found, empty array is returned.

## Description:

A token is a sequence of characters between two (same or different) delimiters (or delimiter group), or between delimiter and begin/end of the string. The Token() function allows you to break down date and time strings, sentences, file names and paths, etc.

TokenArray() is provided for your convenience. It process Token() for all available tokens in <expC1> and returns the resulting token strings in one-dimensional array.

## Example:

```
aToken := TokenArray( "This is FS2-Toolbox." )  
aEval(aToken, {|x| qout(x) } ) // "This" "is" "FS2" "Toolbox"  
  
cStr := "one,two,,four"  
aToken := TokenArray(@cStr)  
aEval(aToken, {|x| qout(x) } ) // "one" "two" "four"  
aToken := TokenArray(cStr, NIL, 1)
```

```

aeval(aToken, {|x| qout(x) } ) // "one" "two" "" "four"

// delimiter groups are '.', ' or ',, ' or ',.' or '..'
aToken := TokenArray(cStr, ".,,.", NIL, 2)
aeval(aToken, {|x| qout(x) } ) // "one,two" "four"

cStr := "one.two.,three....four"
cSep := ".,,"
aToken := TokenArray(cStr, cSep)
aeval(aToken, {|x| qout(x) } ) // "one" "two" "three" "four"

aToken := TokenArray(cStr, cSep, 1)
aeval(aToken, {|x| qout(x) } ) // "one" "two" "" "three" "" ""...

aToken := TokenArray(cStr, cSep, 0, 2)
aeval(aToken, {|x| qout(x) } ) // "one.two" "three" "four"

aToken := TokenArray(cStr, cSep, 1, 2)
aeval(aToken, {|x| qout(x) } ) // "one.two" "three" "" "four"

```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, not available in NT2/CA3 tools.

**Related:**

FS2:TokenArray(), FS2:AtToken(), FS2:NumToken(), FS2:TokenSep(),  
 FS2:TokenUpper(), FS2:TokenLower()

# TOKENAT ()

---

**Syntax:**

**retN = TokenAt ( [expL1] )**

**Purpose:**

Determines the most recent TokenNext() position within a string.

**Options:**

<expL1> is a switch specifying whether the token or delimiter position should be reported. If .F. or NIL (the default), <retN> reports the position of current token within the original string. If .T., the <retN> returns the delimiter position after that token.

**Returns:**

<retN> is a position of the current token or of the delimiter behind this token.

**Description:**

TokenAt() is belonging to the group of versatile incremental tokenizer functions, initialized by TokenInit() and processed by TokenNext().

This function returns the position of the current token recently extracted by TokenNext(), or the position of the trailing delimiter.

To get the delimiter preceeding the current token, use Substr(cString, TokenAt() -1, 1), see example below.

To get the delimiter behind the current token, use Substr(cString, TokenAt(.T.), 1), see example below.

**Example:**

```
cStr := "This is a search,234"
TokenInit(cString)

? TokenNext(), TokenAt(), TokenAt(.T.) // Token = "This"    1    5
? TokenNext(), TokenAt(), TokenAt(.T.) // Token = "is"      6    8
? TokenNext(), TokenAt(), TokenAt(.T.) // Token = "a"        9   10
? TokenNext(), TokenAt(), TokenAt(.T.) // Token = "search" 11   17
? substr(cStr,TokenAt() -1, 1)           // Delim = " "
? substr(cStr,TokenAt(.T.), 1)           // Delim = ","
? TokenNext(), TokenAt(), TokenAt(.T.) // Token = "234"    18   21
? substr(cStr,TokenAt() -1, 1)           // Delim = ","
? substr(cStr,TokenAt(.T.), 1)           // Delim = ""
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:TokenInit(), FS2:TokenNext(), FS2:TokenEnd()

# TOKENEND ()

---

**Syntax:**

```
retL = TokenEnd ( )
```

**Purpose:**

Determines if more tokens are available by TokenNext().

**Returns:**

<retL> is .T. after the last token has been queried with TokenNext() and is .F. otherwise.

**Description:**

TokenEnd() is belonging to the group of versatile incremental tokenizer functions, initialized by TokenInit() and processed by TokenNext().

When TokenNext() is used in FOR..NEXT or WHILE..ENDDO loops, you need to test for a termination condition. TokenEnd() provides such a condition. Note that a null-string is not a safe termination condition, since it may be sometimes returned as a valid token, depending on the <expN3> parameter in TokenInit().

**Example:**

```
cStr := "one, two, ,four,,six,,eight"
TokenInit(cStr)
while !TokenEnd()
    ? TokenNext()      // "one" "two" "four" "six" "eight"
enddo

TokenInit(cStr,,2)
while !TokenEnd()
    ? TokenNext()      // "one" "two" "" "four" "six" "eight"
enddo
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:FS2:TokenInit(), FS2:TokenNext(), FS2:TokenAt()

# TOKENINIT ()

---

## **Syntax:**

**retL = TokenInit ( [expC1], [expC2], [expN3] )**

## **Purpose:**

Initializes a string for TokenNext() processing.

## **Arguments:**

**<expC1>** is the string that is processed. It can be passed by value or by reference. When passed by reference, TokenInit() will modify the <expC1> value, to achieve backward compatibility to CA Tools3.

- If <expC1> is not specified or is NIL, TokenNext() is reset to begin of the string (other parameters are ignored in this case).
- If null-string "" is given, only the memory of internal TokenInit() buffer is cleared.

**<expC2>** is a list of delimiters used to identify the tokens. If not specified, the default <expC2> delimiters are:

" , . ; : ! ? / \ < > ( ) # ^ % & + - \* " + chr(9) + chr(13) + chr(10) + chr(26) + chr(138) + chr(141) + chr(4)

**<expN3>** is a max number of delimiters before empty token is returned. The default value is 0 indicating that empty tokens are not taken into account. A value of 1 will create even empty token for each delimiter found. Value of 2 will create empty token when at least two subsequent delimiters are detected, and so forth.

## **Returns:**

**<retL>** is .T. on success and .F. on failure.

## **Description:**

TokenInit() is belonging to the group of versatile incremental tokenizer functions, which are:

TokenInit(), TokenNext(), TokenEnd(), TokenAt(), SaveToken(), RestToken()

This tokenizer group is designed to process efficiently large strings and is an alternative to stand-alone functions FS2:Token() and FS2:TokenArray().

TokenInit() initializes the tokenizing process for TokenNext(). It stores the <expC1> string internally and replaces all delimiters by one common delimiter to avoid subsequent searches. As opposite to CT3, it is not required in FS2 to pass <expC1> by reference, and therefore also constants or database fields may be used.

Embedded zero bytes in <expC1> and <expC2> are not considered and terminates the string. If required, use Token() or TokenArray() instead.

**Example:**

```
cString := "A.B-C,D!E??"
cDelim  := "!?.,-"

TokenInit(cString, cDelim)
? cString // "A.B-C,D!E??" (= unchanged)
while .not. TokenEnd()
    ? TokenNext() // "A" "B" "C" "D" "E"
enddo

TokenInit(@cString, cDelim)
? cString // "A!B!C!D!E!!" (= changed)

TokenInit("") // you may free the internal buffer
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools requires passing of <expC1> by reference.

**Related to this group:**

FS2:TokenNext(),      FS2:TokenAt(),      FS2:TokenEnd(),      FS2:SaveToken(),  
FS2:RestToken()

**Related:**

FS2:Token(),      FS2:TokenArray(),      FS2:NumToken(),      FS2:AtToken(),  
FS2:TokenLower(), FS2:TokenUpper(), TokenSep()

# TOKENLOWER ()

---

**Syntax:**

**retC = TokenLower ( expC1, [expC2], [expN3] )**

**Purpose:**

Converts the initial alphabetic character of a token into lower case.

**Arguments:**

<expC1> is the string that is processed. Embedded zero bytes are supported.

**Options:**

<expC2> is a list of delimiters used to identify the tokens. If not specified, the default <expC2> delimiters are:

" ,.:;!?\<>()#^%&+-\*" + chr(9) + chr(13) + chr(10) + chr(26) + chr(138) + chr(141) + chr(4) + chr(0)

Embedded zero bytes are supported.

<expN3> is the number of tokens, in which the conversion should occur, i.e. 3 will convert the first character in tokens 1, 2 and 3 but not in 4 etc. The default value is 0 which converts all tokens.

**Returns:**

<retC> is the modified string.

**Description:**

TokenLower() converts the initial alphabetic character of all or specified tokens in the <expC1> into a lower case character. The processed string <expC1> is then returned in <retC>.

**Example:**

```
? TokenLower("Good Morning")           // "good morning"
? TokenLower("Good Morning",, 1)         // "good Morning"

cStr := "AB,AB,AB"
? TokenLower(@cStr, ",", 2)              // "aB,aB,AB"
? cStr                                    // "AB,AB,AB"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:TokenUpper(),    FS2:Token(),    FS2:TokenArray(),    FS2:NumToken(),  
FS2:AtToken(), Lower()



# TOKENNEXT ()

---

**Syntax:**

```
retC = TokenNext ( [expC1] )
```

**Purpose:**

Provides an incremental tokenizer.

**Options:**

<expC1> is an optional string previously initialized by TokenInit(). It is not used and not required in FS2 but supported for backward CT3 compatibility.

**Returns:**

<retC> is a string containing the next token. If no more tokens are available, null-string "" is returned.

**Description:**

TokenNext() is belonging to the group of versatile incremental tokenizer functions, initialized by TokenInit() and processed by TokenNext(). Generally speaking, it is a speed optimized version of FS2:Token() or FS2:TokenArray() and used mainly for large strings.

When TokenNext() is used in FOR..NEXT or WHILE..ENDDO loops, you need to test for a termination condition. TokenEnd() provides such a condition. Note that a null-string is not a safe termination condition, since it may be sometimes returned as a valid token, depending on the <expN3> parameter in TokenInit().

As opposite to CA Tools3, FS2 TokenNext() does not need the original string, since TokenInit() stores it internally. This avoids problems and break-downs known from Clipper. If you are very short with memory and/or are using huge strings, you may free the internal buffer after processing all TokenNext() by invoking TokenInit("").

Embedded zero bytes in are not considered by TokenNext(). If such are required, use Token() or TokenArray() instead.

**Example:**

```
cStr := "what a beautiful day today"
TokenInit(cStr)
while .not. TokenEnd()
    ? TokenNext()           // "what" "a" "beautiful" "day" "today"
enddo

cStr := "one, two,,four,,six,,eight"
TokenInit(cStr)
while .not. TokenEnd()
    ? TokenNext()           // "one" "two" "four" "six" "eight"
enddo

TokenInit(cStr,,1)
while .not. TokenEnd()
    ? TokenNext()           // "one" "" "two" "" "four" "" "six" "" "eight"
enddo
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related to this group:**

FS2:TokenInit(),      FS2:TokenAt(),      FS2:TokenEnd(),      FS2:SaveToken(),  
FS2:RestToken()

**Related:**

FS2:Token(),      FS2:TokenArray(),      FS2:NumToken(),      FS2:AtToken(),  
FS2:TokenLower(), FS2:TokenUpper(), TokenSep()

# TOKENSEP ()

---

**Syntax:**

**retC** = TokenSep ([**expL1**] )

**Purpose:**

Provides the separator before or after the token most recently retrieved by Token().

**Options:**

<**expL1**> is the required mode. When specified .T., the function returns the delimiter after the most recently retrieved token. The default value (.F.) specifies that the delimiter before the token most recently retrieved is returned.

**Returns:**

<**retC**> is the delimiter found directly before the token most recently retrieved. If there is no delimiting character present at the beginning or end of a character string, the function returns a null string.

**Description:**

Sometimes are the delimiting characters of great interest, e.g. when mathematical expressions are broken down. TokenSep() is always concerned with the most recently retrieved token determined by the Token() function. Using the <expL1> parameter, you can specify whether the delimiter returned is the one before or after the token most recently retrieved.

**Example:**

```
? Token("Hello,world!")      // Last token      : "world"
? TokenSep()                  // Leading separator : ","
? TokenSep(.T.)               // Trailing separator: "!"

? Token("32 + 45 * 70", "+-*/", 2)      // "45"
? TokenSep(), TokenSep(.T.)            // "+" "*"

? TokenSep() + (Token("32+45*70", "+-*/",2)) // "+45"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:Token(),    FS2:FS2:NumToken(),    FS2:AtToken(),    FS2:TokenLower(),  
FS2:TokenUpper(), FS2:TokenSepDef()

# TOKENSEPDEF ( )

---

**Syntax:**

**retC** = TokenSepDef ( )

**Purpose:**

Return the list of default separators.

**Returns:**

<**retC**> is the list of default separators.

**Description:**

You may use TokenDefSep() when testing manually the separators by TokenSep(), or TokenAt() functions.

**Example:**

```
? TokenSep()  
  // " , . ; : ! ? \ / < > ( ) # ^ % & + - * " + chr(9) + chr(13) + chr(10) + ;  
  // chr(26) + chr(138) + chr(141) + chr(4) + chr(0)  
  
? Token("Hello,world!",,1)  
cSep := TokenSep()  
? "'" + cSep + "'" is "  
if !(cSep $ TokenSep())  
  ?? "not "  
endif  
?? "default token separator"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, not available in NT2/CA3 tools.

**Related:**

FS2:Token(), FS2:TokenAt(), FS2:TokenSep()

# TOKENUPPER ()

---

**Syntax:**

```
retC = TokenUpper ( expC1, [expC2], [expN3] )
```

**Purpose:**

Converts the initial alphabetic character of a token into upper case.

**Arguments:**

<expC1> is the string that is processed. Embedded zero bytes are supported.

**Options:**

<expC2> is a list of delimiters used to identify the tokens. If not specified, the default <expC2> delimiters are:

```
" ,.;:!?/\<>()#^%&+-*" + chr(9) + chr(13) + chr(10) + chr(26) +  
chr(138) + chr(141) + chr(4) + chr(0)
```

Embedded zero bytes are supported.

<expN3> is the number of tokens, in which the conversion should occur, i.e. 3 will convert the first character in tokens 1, 2 and 3 but not in 4 etc. The default value is 0 which converts all tokens.

**Returns:**

<retC> is the modified string.

**Description:**

TokenUpper() converts the initial alphabetic character of all or specified tokens in the <expC1> into a upper case character. The processed string <expC1> is then returned in <retC>.

**Example:**

```
? TokenUpper("good morning")           // "Good Morning"  
? TokenUpper("good morning",, 1)        // "Good morning"  
  
cStr := "ab,ab,ab"  
? TokenUpper(@cStr, ",", 2)             // "Ab,Ab,ab"  
? cStr                                   // "ab,ab,ab"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:TokenLower(),    FS2:Token(),    FS2:TokenArray(),    FS2:NumToken(),  
FS2:AtToken(), Upper()

# VALPOS ()

---

**Syntax:**

**retN = ValPos ( expC1, [expN2] )**

**Purpose:**

Determines the numerical value of the character at a particular position.

**Arguments:**

<expC1> is the string that is processed. Embedded zero bytes are supported.

**Options:**

<expN2> is the character position in <expC1> which should be converted into numeric data format. The default value processes the last character, i.e. is equivalent to passing len(expC1)

**Returns:**

<retN> is the numeric value of the character at position <expN2>. If the character is not numeric or out of the <expC1> range, 0 is returned.

**Description:**

ValPos() is similar to the standard Val() function. While Val() returns a number representing all subsequent numeric characters starting at string begin, ValPos() return the number of one single character at specified position.

**Example:**

```
cString := "AX412B"
? ValPos(cString, 2)      // 0
? ValPos(cString, 3)      // 4

? Val(substr(cString, 3)) // 412
? Asc(substr(cString, 3)) // 52
? Strpeek(cString, 3)     // 52
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:AscPos(), Val(), Asc(), Strpeek()

# WORDONE ( )

---

**Syntax:**

**retC = WordOne ( [expC1], expC2 )**

**Purpose:**

Reduces multiple appearances of particular double characters to one.

**Arguments:**

<expC1> is the sequence of two characters which should appear only once together in <expC2>. The default value is for all 2-byte sequences.

<expC1> is the string that is processed.

Note: If only one parameter is given, a parameter shift is assumed and <expC1> is treated as <expC2>. Such a parameter shift is a bad programming technique and not suggested, but handled by FS2 for compatibility purposes to CT3.

**Returns:**

<retC> is the modified string.

**Description:**

A unique operation is carried out on a string that is constructed out of 2-byte sequences ("words"). The multiple sequence appearances must lie immediately beside one another, which allows a CharSort() with an element length of 2 to be executed.

These sequences can be integers that have been generated using the I2BIN() function and have been deposited in a string. In conjunction with other string functions like WordOnly(), WordOne() is an extremely effective system for working with these kinds of files.

The term "word" is not used here in the textual sense, but rather as it is used in C or assembler programming. A "word" in this context consists of units of 2 bytes.

**Example:**

```
// This is a simple example with characters that can be
// displayed. The "AB" lie one after the other but not the "12":

? wordOne("12ABAB12")           // "12AB12"

// The function always runs through the string in ordered pairs:

? wordOne("12", "1212ABAB")      // "12ABAB"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:CharOne()

# WORDONLY ()

---

**Syntax:**

```
retC = WordOnly ( expC1, expC2 )
```

**Purpose:**

Finds the common denominator between two strings on the basis of double characters.

**Arguments:**

<expC1> is a the 2-byte sequences that are not removed from <expC2>.

<expC2> is the processed string from which 2-byte sequences that do not appear in <expC1> are removed

**Returns:**

<retC> is the modified string.

**Description:**

WordOnly() reports all 2-byte sequences in <expC2> that appear in <expC1>. This function is very useful in applications where 16-bit integer values have been saved to strings using the I2BIN() function. The WordOnly() function always processes strings in ordered pairs.

The term "word" is not used here in the textual sense, but rather as it is used in C or assembler programming. A "word" in this context consists of units of 2 bytes.

**Example:**

```
? wordOnly("AABBCCDD", "XXAAYYBBZZ") // "AABB"
? wordOnly("BBA", "XXAAYYBBAZ") // "BB"
? wordOnly("BBBA", "XXAAYYBBBAZ") // "BBBA"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:CharOnly()



# WORDREPL ()

---

**Syntax:**

```
retC = WordRepl ( expC1, expC2, expC3, [expL4] )
```

**Purpose:**

Replaces particular double characters with others.

**Arguments:**

<expC1> is a string containing sequence of multiple 2-byte character which are replaced by corresponding characters from <expC3> within the <expC2> string.

<expC2> is the string that is processed.

<expC3> is a string containing sequence of multiple 2-byte character that replaces <expC1> in the <expC2> string

**Options:**

<expL4> specifies whether the character string <expC2> is run through in single steps .T., or in ordered pairs .F. (default).

**Returns:**

<retC> is the modified string.

**Description:**

WordRepl() exchange all 2-byte sequences in the <expC2> string for another sequence of the same length. By using WordRepl() and its analog function CharRepl(), you can develop a very fast algorithm for soundex() functions.

The <expC2> can be processed in a number of different ways. It is here that the <expL4> parameter plays an important role, similar to the CsetAtmupa() functions.

There are different modes of execution for the function:

- Depending on the <expL4> parameter, the string is processed in single steps or ordered pairs.

- CsetAtmupa() is successful only when <xpL4> is .T. If CsetAtmupa() is .T., then WordRepl() continues with the second character exchanged in a sequence. If CsetAtmupa() is .F., WordRepl() continues with the character behind the second character.

The term "word" is not used here in the textual sense, but rather as it is used in C or assembler programming. A "word" in this context consists of units of 2 bytes.

**Example:**

```
? WordRepl("CC", "AABBCCDDEE", "XX")           // "AABBXXDDEE"

CsetAtmupa(.F.)                                   // Default
? WordRepl("aa", "1aaaa", "ba")                  // "1abaa"
? WordRepl("aa", "1aaaa", "ba", .T.)             // "1baba"

CsetAtmupa(.T.)                                   // Multi-pass mode on
? WordRepl("aa", "1aaaa", "ba")                  // "1abaa"
? WordRepl("aa", "1aaaa", "ba", .T.)             // "1bbba"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:CharRepl(), FS2:CsetAtmupa()

# WORDSWAP ()

---

**Syntax:**

```
retC = WordSwap ( expC1, [expL2] )
```

**Purpose:**

Exchanges double characters that lie beside each other in a string.

**Arguments:**

<expC1> is the string that is processed. Embedded zero bytes are supported.

**Options:**

<expL2> specifies whether the adjoining bytes are exchanged (.T.) or not (.F.), which is the default.

**Returns:**

<retC> is the modified string.

**Description:**

WordSwap() exchange 2-byte sequences within a <expC1> string for each other. In this respect the function works exactly like CharSwap(), except that 16-bit groups are being exchanged.

The function is used on strings that contain 32-bit integers created by L2BIN(). The exchange serves as a preparation for a sort of these integers using CharSort(). In order to achieve a correct sort, you must exchange two adjacent bytes (low/high ordering of 16-bit integers), in addition to exchanging the 16-bit groups.

When <expL2> is .T., adjoining bytes are only exchanged after being exchanged as two 2-byte groups.

The term "word" is not used here in the textual sense, but rather as it is used in C or assembler programming. A "word" in this context consists of units of 2 bytes.

**Example:**

```
? WordSwap("1234567890")           // "3412785690"
? WordSwap("1234567890", .T.)       // "4321876590"

? L2BIN(65536)                       // chr(0) + chr(0) + chr(1) + chr(0)
? L2BIN(1)                           // chr(1) + chr(0) + chr(0) + chr(0)

? WordSwap(L2BIN(65536), .T.)        // chr(0) + chr(1) + chr(0) + chr(0)
? WordSwap(L2BIN(1), .T.)            // chr(0) + chr(0) + chr(0) + chr(1)

? L2BIN(65536) > L2BIN(1)             // .F.
? WordSwap(L2BIN(65536), .T.) > WordSwap(L2BIN(1), .T.) // .T.
```

**Classification:** add-on library, programming

**Compatibility:** Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only the first optional argument.

**Related:** FS2:ChrSwap(), ChrSort()

# WORDTOCHAR ()

---

**Syntax:**

```
retC = WordToChar( expC1, expC2, expC3 )
```

**Purpose:**

Exchanges double characters for individual ones.

**Arguments:**

<expC1> is a string containing sequence of multiple 2-byte character that will be replaced by corresponding <expC3> in the <expC2> string

<expC2> is the string that is processed.

<expC3> is a string containing sequence of single characters which correspond to 2-byte sequences in <expC1> string.

**Returns:**

<retC> is the modified string.

**Description:**

When you use SOUNDEX algorithms, sequences of two characters must often be exchanged for a single other character. WordToChar() makes this process extremely simple and quick.

The function processes the <expC2> in 1-byte steps. The behavior after exchanging a sequence for a character is determined by CsetAtmupa(). If this is set .F. (the default), the search for more sequences continues after the exchanged characters. If CsetAtmupa() is .T., the search for more sequences continues and includes the exchanged characters.

The <expC3> string can be shorter than <expC1>. When this occurs, the function exchanges the sequences in <expC1> that do not have corresponding sequences in <expC3> for the last sequence in <expC1>.

The term "word" is not used here in the textual sense, but rather as it is used in C or assembler programming. A "word" in this context consists of units of 2 bytes.

**Example:**

```
? wordToChar("aa", "xaaaa", "a")      // "xaa"
CsetAtmupa(.F.)                          // Multi pass off
? wordToChar("aa", "xaaaa", "a")      // "xaa"
CsetAtmupa(.T.)                          // Multi pass on
? wordToChar("aa", "xaaaa", "a")      // "xa"
```

**Classification:** add-on library, programming

**Compatibility:** Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:** FS2:WordRepl(), FS2:CharRepl(), FS2:CsetAtmupa()

# Numeric, Mathematical Functions

---

## Introduction

The FS2 Toolbox **Mathematical functions** includes trigonometric functions, complex finance-oriented math computations, and functions to determine the factorial, sign, or the next-largest integer of a value.

The FS2 Toolbox **Number and Bit Manipulation** provides several number and bit manipulations, e.g. conversions between two different number systems, creation of random numbers, binary operations as AND, OR, XOR, NOT, and functions to test, set and clear bits.

## Index of FS2 Mathematical Functions

ACOS()	Computes the cosine arc
ASIN()	Computes the sine arc
ATAN()	Computes the tangent arc
ATN2()	Computes the angle size from the sine and cosine
CEILING()	Rounds up to the next integer
COS()	Computes the cosine
COT()	Computes the cotangent
DTOR()	Converts from a degree to radian measure
FACT()	Computes the factorial
FLOOR()	Rounds down to the next integer
FV()	Computes future value of capital
GETPREC()	Determines the level of precision that is set
LOG10()	Computes the common logarithm
PAYMENT()	Computes the periodic payment amount
PERIODS()	Computes number of payment periods necessary to repay a loan
PI()	Returns pi with the highest degree of accuracy
PV()	Computes the cash present value after interest charges
RATE()	Computes the interest rate for a loan
RTOD()	Converts from a radian to degree measure
SETPREC()	Sets the precision for trigonometric functions (CT3 only)
SIGN()	Determines the mathematical sign of a number
SIN()	Computes the sine of a radian value
TAN()	Computes the tangent of a radian value

## Index of FS2 Number and Bit Manipulation

BITTOC()	Converts position-dependent bits into characters
CELSIUS()	Converts a Fahrenheit temperature value into Celsius
CLEARBIT()	Clears one or more bits within a number to zero
CTOBIT()	Converts a character string into a bit pattern

CTOF()	Converts a special 8-byte string into a floating point number
CTON()	Converts a numeric string into a different base
EXPONENT()	Determines the exponent of a floating point number (base 2)
FAHRENHEIT()	Converts a temperature value from Celsius into Fahrenheit
FTOC()	Converts a floating point number into a special 8-byte string
INFINITY()	Creates the largest number possible (21023)
INTNEG()	Converts an unsigned integer into a signed integer
INTPOS()	Converts a signed integer into an unsigned integer
ISBIT()	Tests the bits in a number
LTON()	Converts a logical value into a numeric value
MANTISSA()	Determines the mantissa of a floating point number (base2)
NTOC()	Converts numbers in a string into a different number base
NUMAND()	Performs a 16-bit "AND" of a list of numbers
NUMCOUNT()	Increment or set an internal counter
NUMHIGH()	Returns the higher value byte in a 16-bit number
NUMLOW()	Returns the lower value byte in a 16-bit number
NUMMIRR()	Mirrors 8-bit or 16-bit values
NUMNOT()	Performs a 16-bit "NOT" of a number
NUMOR()	Performs a 16-bit "OR" of a list of numbers
NUMROL()	Performs a 16-bit left rotation of a number
NUMXOR()	Performs a 16-bit "XOR" of two numbers
RAND()	Generates random numbers
RANDOM()	Generates random numbers
SETBIT()	Sets one or more bits in a number

# Mathematical Functions

---

The FS2 Toolbox Mathematical functions includes trigonometric functions, complex finance-oriented math computations, and functions to determine the factorial, sign, or the next-largest integer of a value.

# ACOS ()

---

**Syntax:**

**retN = Acos ( expN1 )**

**Purpose:**

Computes the cosine arc.

**Arguments:**

<expN1> is an angle in radians. The value is in the range of -1.00 to +1.00 inclusive). If you want to specify the value in degrees, use DtoR(expN1) conversion.

**Returns:**

<retN> is the result of computation acos(expN1) or -999.99 on error

**Description:**

Acos() computes an angle size in radians for a cosine value. The returned value is in the range of 0.0 to PI().

Note: the precision in FS2 is same as in FlagShip: double floating point, which is on 32-bit systems in the range - 4.94\*10<sup>-324</sup> to 1.79\*10<sup>308</sup> with usually 15 significant digits. The displayed result by using the ?, ?? or @..SAY etc. commands depends on the precision set by SET DECIMAL and SET FIXED. In doubt, use the str() function to format the numeric value with required decimal digits.

**Example:**

```
nValue := acos(0.7)
? str(nValue, 18, 15)      // 0.795398830184144

nValue := acos(0.5)
? nValue                  // 1.05
? str(nValue, 18, 15)      // 1.047197551196598

SET DECIMALS TO 10
? acos(0.5)                //      1.0471975512
? acos(1.5)                // -999.99000000000
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:Cos(), FS2:Sin(), FS2:Asin(), FS2:Tan(), FS2:Atan(), FS2:Atn2(), FS2:Cot(), FS2:Dtor(), FS2:Rtod()



# ASIN ()

---

**Syntax:**

**retN = Asin ( expN1 )**

**Purpose:**

Computes the sine arc.

**Arguments:**

<expN1> is an angle in radians. The value is in the range of -1.00 to +1.00 inclusive). If you want to specify the value in degrees, use DtoR(expN1) conversion.

**Returns:**

<retN> is the result of computation asin(expN1) or -999.99 on error

**Description:**

Asin() computes an angle size in radians for a sine value. The returned value is in the range of -PI()/2 to PI()/2.

Note: the precision in FS2 is same as in FlagShip: double floating point, which is on 32-bit systems in the range - 4.94\*10<sup>-324</sup> to 1.79\*10<sup>308</sup> with usually 15 significant digits. The displayed result by using the ?, ?? or @..SAY etc. commands depends on the precision set by SET DECIMAL and SET FIXED. In doubt, use the str() function to format the numeric value with required decimal digits.

**Example:**

```
nValue := asin(0.7)
? str(nValue, 18, 15)      // 0.775397496610753

nValue := asin(0.5)
? nValue                  // 0.52
? str(nValue, 18, 15)      // 0.523598775598299

SET DECIMALS TO 10
? asin(0.5)                //      0.5235987756
? asin(1.5)                // -999.99000000000
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:Cos(), FS2:Sin(), FS2:Acos(), FS2:Tan(), FS2:Atan(), FS2:Atn2(), FS2:Cot(),  
FS2:Dtor(), FS2:Rtod()

# ATAN ()

---

**Syntax:**

**retN = Atan ( expN1 )**

**Purpose:**

Computes the tangent arc.

**Arguments:**

<expN1> is an angle in radians. If you want to specify the value in degrees, use DtoR(expN1) conversion.

**Returns:**

<retN> is the result of computation atan(expN1) or -999.99 on error

**Description:**

Atan() computes an angle size in radians for a tangent value.

Note: the precision in FS2 is same as in FlagShip: double floating point, which is on 32-bit systems in the range - 4.94\*10<sup>-324</sup> to 1.79\*10<sup>308</sup> with usually 15 significant digits. The displayed result by using the ?, ?? or @..SAY etc. commands depends on the precision set by SET DECIMAL and SET FIXED. In doubt, use the str() function to format the numeric value with required decimal digits.

**Example:**

```
nValue := atan(PI()/4)
? str(nValue, 18, 15)      // 0.665773750028354

nValue := atan(0.5)
? nValue                  // 0.46
? str(nValue, 18, 15)      // 0.463647609000806

SET DECIMALS TO 10
? atan(0.5)                // 0.4636476090
? atan(1500)               // 1.5701296602
? atan()                   // -999.9900000000
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:Atn2(), FS2:Cos(), FS2:Sin(), FS2:Acos(), FS2:Asin(), FS2:Tan(), FS2:Atn2(), FS2:Cot(), FS2:Dtor(), FS2:Rtod()

# ATN2 ()

---

**Syntax:**

**retN = Atn2 ( expN1, expN2 )**

**Purpose:**

Computes the angle size from the sine and cosine.

**Arguments:**

<expN1> is the sine value for an angle. If you want to specify the value in degrees, use the DtoR(expN1) conversion.

<expN2> is the cosine value for an angle. If you want the values specified in degrees, use DtoR(expN2) conversion.

**Returns:**

<retN> is the angle sine in radians or -999.99 on error

**Description:**

Atn2() returns the angle sine in radians, where the sine and the cosine of a given point have been specified. The function returns results for all four quadrants and corresponds to a call of ATAN(x/y). One advantage of the ATN2() function is that no "divide by zero" error can occur. The returned value is in the range of -PI() to PI().

Note: the precision in FS2 is same as in FlagShip: double floating point, which is on 32-bit systems in the range - 4.94\*10<sup>-324</sup> to 1.79\*10<sup>308</sup> with usually 15 significant digits. The displayed result by using the ?, ?? or @..SAY etc. commands depends on the precision set by SET DECIMAL and SET FIXED. In doubt, use the str() function to format the numeric value with required decimal digits.

**Example:**

```
nSin := sin(DtoR(30))      // 30 degree in radians
nCos := cos(DtoR(30))      // 30 degree in radians

SET DECIMALS TO 6
nVal := atn2(nSin, nCos)
? nVal                     // 0.523599 radians
? RtoD(nVal)               // 30.000000 degrees
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:Atan(), FS2:Cos(), FS2:Sin(), FS2:Acos(), FS2:Asin(), FS2:Tan(), FS2:Cot(),  
FS2:DtoR(), FS2:RtoD()

# CEILING ()

---

**Syntax:**

`retN = Ceiling ( expN1 )`

**Purpose:**

Rounds up to the next integer.

**Arguments:**

<expN1> is the number that is processed.

**Returns:**

<retN> is the next-largest integer to the <expN1> value.

**Description:**

Ceiling() returns the next-largest integer to the one passed as a parameter. This applies to positive and negative numbers.

Note: the precision in FS2 is same as in FlagShip: double floating point, which is on 32-bit systems in the range -  $4.94 \cdot 10^{-324}$  to  $1.79 \cdot 10^{308}$  with usually 15 significant digits. The displayed result by using the ?, ?? or @..SAY etc. commands depends on the precision set by SET DECIMAL and SET FIXED. In doubt, use the str() function to format the numeric value with required decimal digits.

**Example:**

```
? Ceiling(1.9)           // 2
? Ceiling(1.1)           // 2
? Ceiling(0.9)           // 1
? Ceiling(-0.1)          // 0
? Ceiling(-0.9)          // 0
? Ceiling(-1.1)          // -1
? Ceiling(1234567890123.456) // 1234567890124.00
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:Floor()

# COS ( )

---

**Syntax:**

**retN = Cos ( expN1 )**

**Purpose:**

Computes the cosine.

**Arguments:**

<expN1> is an angle in radians. If you want to specify the value in degrees, use DtoR(expN1) conversion.

**Returns:**

<retN> is the result of computation cos(expN1) or -999.99 on error

**Description:**

Cos() computes cosine value for a value in radians.

Note: the precision in FS2 is same as in FlagShip: double floating point, which is on 32-bit systems in the range - 4.94\*10<sup>-324</sup> to 1.79\*10<sup>308</sup> with usually 15 significant digits. The displayed result by using the ?, ?? or @..SAY etc. commands depends on the precision set by SET DECIMAL and SET FIXED. In doubt, use the str() function to format the numeric value with required decimal digits.

**Example:**

```
nValue := cos(0.7)
? str(nValue, 18, 15)           // 0.764842187284488

nValue := cos(0.5)
? nValue                       // 0.88
? str(nValue, 18, 15)           // 0.877582561890373

SET DECIMALS TO 10
? cos(PI())                     // -1.0000000000
? cos(PI() * 99)                // -1.0000000000

? cos(DtoR(30))                 // 0.8660254038
? cos(DtoR(90))                 // 0.0000000000
? cos(DtoR(180))                // -1.0000000000
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:Acos(), FS2:Sin(), FS2:Asin(), FS2:Tan(), FS2:Atan(), FS2:Atn2(), FS2:Cot(), FS2:DtoR(), FS2:RtoD()

# COT ()

---

**Syntax:**

**retN = Cot ( expN1 )**

**Purpose:**

Computes the cotangent.

**Arguments:**

<expN1> is an angle in radians. If you want to specify the value in degrees, use DtoR(expN1) conversion.

**Returns:**

<retN> is the result of computation cot(expN1) or -999.99 on error

**Description:**

Cot() computes the cotangent value for a value in radians.

Note: the precision in FS2 is same as in FlagShip: double floating point, which is on 32-bit systems in the range - 4.94\*10<sup>-324</sup> to 1.79\*10<sup>308</sup> with usually 15 significant digits. The displayed result by using the ?, ?? or @..SAY etc. commands depends on the precision set by SET DECIMAL and SET FIXED. In doubt, use the str() function to format the numeric value with required decimal digits.

**Example:**

```
nValue := cot(0.7)
? str(nValue, 18, 15)           // 1.187241832126679

nValue := cot(0.5)
? nValue                       // 1.83
? str(nValue, 18, 15)           // 1.830487721712452

SET DECIMALS TO 10
? cot(0.5)                     // 1.8304877217
? cot(PI())                    // 1305964182209525.7500000000
? cot(PI() * 99)               // 14370126108396.7910156250

? cot(DtoR(0))                 // -999.99000000000
? cot(DtoR(30))                // 1.7320508076
? cot(DtoR(90))                // 0.0000000000
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:Cos(), FS2:Asin(), FS2:Acos(), FS2:Tan(), FS2:Atan(), FS2:Atn2(), FS2:Sin(), FS2:DtoR(), FS2:RtoD()

# DTOR ( )

---

**Syntax:**

`retN = DtoR ( expN1 )`

**Purpose:**

Converts degree to radians.

**Arguments:**

<expN1> is a valid angle measurement in degrees.

**Returns:**

<retN> is the result in radians or -999.99 on error

**Description:**

With DtoR(), you can convert degree measurements into radians.

Note: the precision in FS2 is same as in FlagShip: double floating point, which is on 32-bit systems in the range -  $4.94 \times 10^{-324}$  to  $1.79 \times 10^{308}$  with usually 15 significant digits. The displayed result by using the ?, ?? or @..SAY etc. commands depends on the precision set by SET DECIMAL and SET FIXED. In doubt, use the str() function to format the numeric value with required decimal digits.

**Example:**

```
? str(DtoR(360), 18, 15)    // 6.283185307179588
? str(DtoR(180), 18, 15)    // 3.141592653589794
? str(DtoR(180.5), 18, 15)  // 3.150319299849766
SET DECIMALS TO 10
? DtoR(720)                 // 12.5663706144
? DtoR(-180)               // -3.1415926536
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:RtoD()

# FACT ()

---

**Syntax:**

**retN = FACT ( expN1 )**

**Purpose:**

Computes the factorial.

**Arguments:**

<expN1> is the computed value

**Returns:**

<retN> is the result of the computation or -999.99 on error

**Description:**

Fact() computes the factorial of the given value.

Note: the precision in FS2 is same as in FlagShip: double floating point, which is on 32-bit systems in the range -  $4.94 \times 10^{-324}$  to  $1.79 \times 10^{308}$  with usually 15 significant digits. The displayed result by using the ?, ?? or @..SAY etc. commands depends on the precision set by SET DECIMAL and SET FIXED. In doubt, use the str() function to format the numeric value with required decimal digits.

**Example:**

```
? Fact(1)           // 1
? Fact(5)           // 120
? Fact(21)          // 51090942171709440000
? Fact(25)          // 15511210043330986055303168
? Fact(100)         // 9.332621544394410218832560 * 10**157
? Fact(0)           // -999.99
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only values 0 to 21.



# FLOOR ()

---

**Syntax:**

**retN = Floor ( expN1 )**

**Purpose:**

Rounds down to the next integer.

**Arguments:**

<expN1> is the number for which the next-smaller integer is determined

**Returns:**

<retN> is the next-smaller integer of the <expN1> value

**Description:**

Floor() always returns the next-smaller integer of the one passed as a parameter. This applies to positive and negative numbers.

**Example:**

```
? Floor(1.9)           // 1
? Floor(1.1)           // 1
? Floor(0.9)           // 0
? Floor(-0.1)          // -1
? Floor(-0.9)          // -1
? Floor(-1.1)          // -2
? Floor(12345678901234.56) // 12345678901234.00
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:Ceiling()

# FV ()

---

**Syntax:**

**retN = FV ( expN1, expN2, expN3 )**

**Purpose:**

Computes future value of capital.

**Arguments:**

<expN1> is the installment amount to pay for the period

<expN2> is the interest rate calculated for the payment period. 1.0 corresponds to 100%, 0.05 corresponds to 5% and so forth.

<expN3> is the number of payments

**Returns:**

<retN> is the future capital value of the total deposits, plus the interest generated, or -999.99 on error

**Description:**

FV() returns the capital (future value) available after the <expN3> total installments at <nInstall> payment, at an <expN2> interest charge for the period. The FV() formula is

$$C = R * \frac{q^n - 1}{q - 1}$$

where C = capital, R = rate per period, q = interest factor, n = number of periods

**Example:**

```
// what amount would you save, if you pay $150.00 per month for
// 3 years into a fund that pays an annual interest rate of 6%?

nRate := 0.06/12           // Monthly interest rate of annual 6%
? FV(150, nRate, 36)       // you save $ 5900.42

// what amount would you save, if you pay $200.00 per month for
// 4.5 years into a fund that pays an annual interest rate of 5%?
? FV(200, 0.05/12, 54)    // 12083.40
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:Pv(), FS2:Payment(), FS2:Rate(), FS2:Periods()

# GETPREC ( )

---

**Syntax:**

`retN = GetPrec ( )`

**Purpose:**

Determines the level of precision that is set.

**Returns:**

<retN> is the currently set CA Tools3 precision for trigonometric functions.

**Description:**

This function is supported in FS2 for compatibility purposes to CT3 only. In FS2, the precision is same as in FlagShip: double floating point, which is on 32-bit systems in the range -  $4.94 \times 10^{-324}$  to  $1.79 \times 10^{308}$  with usually 15 significant digits. The displayed result by using the `?`, `??` or `@..SAY` etc. commands depends on the precision set by SET DECIMAL and SET FIXED. In doubt, use the `str()` function to format the numeric value with required decimal digits.

**Example:**

```
? GetPrec() // e.g. 10
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:SetPrec()

# LOG10 ( )

---

**Syntax:**

**retN = Log10 ( expN1 )**

**Purpose:**

Computes the common logarithm.

**Arguments:**

<expN1> is the number for which the base 10 logarithm is determined. The valid range is 0 to  $1.79 \times 10^{308}$ .

**Returns:**

<retN> is the result of  $\log_{10}(\text{expN1})$  or -999.99 on error

**Description:**

Log10() determines the common logarithm for a number. You may see it this way: to what power must 10 be raised ( $10^n$ ) so that <expN1> results?

Note: the precision in FS2 is same as in FlagShip: double floating point, which is on 32-bit systems in the range  $-4.94 \times 10^{-324}$  to  $1.79 \times 10^{308}$  with usually 15 significant digits. The displayed result by using the ?, ?? or @..SAY etc. commands depends on the precision set by SET DECIMAL and SET FIXED. In doubt, use the str() function to format the numeric value with required decimal digits.

**Example:**

```
? Log10(0.01)           // -2.00
? Log10(2)              //  0.30
? Log10(100)            //  2.00
? Log10(-2)             // -999.99
SET DECIMALS TO 15
? Log10(Infinity())     // 308.254715559916747
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

Log()

# PAYMENT ()

---

**Syntax:**

**retN = Payment ( expN1, expN2, expN3 )**

**Purpose:**

Computes the periodic payment amount.

**Arguments:**

<expN1> is the loan amount

<expN2> is the periodic interest rate. 1.0 corresponds to 100%, 0.05 corresponds to 5% and so forth.

<expN3> is the number of payments periods.

**Returns:**

<retN> is the payment to make for each payment period, or -999.99 on error

**Description:**

Payment() computes the payment for each period where loan interest applies. The determined amount relates to a loan amount <expN1> that is re-paid over <expN3>, at an interest rate of <expN2>. The Payment() formula is:

$$R = C * \frac{q - 1}{n} * q^n$$

where C = capital, R = rate per period, q = interest factor, n = number of periods

**Example:**

```
// How high does the monthly annuity amount have to be, if you want
// to repay a $2,000.00 loan within 2 years at an annual interest
// rate of 10%:

nRate := 0.1/12           // The monthly interest rate
? Payment(2000, nRate, 24) // 92.29 (payment per month)
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:FV(), FS2:PV(), FS2:Rate(), FS2:Periods()

# PERIODS ()

---

**Syntax:**

**retN = Periods ( expN1, expN2, expN3 )**

**Purpose:**

Computes the number of payment periods necessary to repay a loan.

**Arguments:**

<expN1> is the loan amount.

<expN2> is the amount of the scheduled periodic payment.

<expN3> is the interest rate calculated for the payment period. 1.0 corresponds to 100%, 0.05 corresponds to 5% and so forth.

**Returns:**

<retN> is the number of payments required to re-pay the <expN1> amount, or -999.99 on error

**Description:**

Periods() computes how often you must make a payment <expN2> to reach the <expN1> amount at the <expN3> interest rate. The Periods() is calculated according to Newton's iterative formula:

$$\{\text{Sum}\} \quad n - \frac{q^{n+1} - (s+1)q^n + s}{\ln(q) * q^n * (q - (s+1))} \quad \text{and} \quad s = \frac{R}{C}$$

where C = capital, R = rate per period, q = interest factor, n = number of periods

**Example:**

```
// How many months do you need to pay back a $4000 loan at an
// annual interest rate of 9.5%, if you want the $200 monthly
// payments?

nRate := 0.095/12           // Monthly interest rate
? Periods(4000, 200, nRate)  // 21.88 months

// Same as above, but the monthly payment of $20 is too small and
// would result in infinite payments:

? Periods(4000, 20, 0.095/12) // -999.99 (error)
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. The CT3 error value is -1.

**Related:**

FS2:FV(), FS2:Pv(), FS2:Payment(), FS2:Rate()

# PI ( )

---

**Syntax:**

**retN = PI ( )**

**Purpose:**

Returns pi with the highest degree of accuracy.

**Returns:**

<retN> is the pi value with the highest degree of accuracy

**Description:**

This function simplifies calculations when the most accurate pi value is required.

**Example:**

```
? str(PI(), 18, 15) // 3.141592653589794
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:Sin(), FS2:Cos(), FS2:Tan(), FS2:Asin(), FS2:Acos(), FS2:Atan()

# PV ()

---

**Syntax:**

**retN = PV ( expN1, expN2, expN3 )**

**Purpose:**

Computes the cash present value after interest charges.

**Arguments:**

<expN1> is the amount of the scheduled periodic payment.

<expN2> is the interest rate calculated for the payment period. 1.0 corresponds to 100%, 0.05 corresponds to 5% and so forth.

<expN3> is the number of anticipated payment periods

**Returns:**

<retN> is the cash value of an interest yield, or -999.99 on error

**Description:**

PV() calculates the cash value (present value) of regular equal payments <expN1> at an <expN2> interest rate over <expN3> payment periods. The PV() formula is

$$C = R * \frac{1}{q} * \frac{q^n - 1}{q - 1}$$

where C = capital, R = rate per period, q = interest factor, n = number of periods

**Example:**

```
// How high can a loan be if you pay $175 for 24 months, at an
// annual fixed interest rate of 9.5%

nRate := 0.095/12           // monthly interest rate
? PV(175, nRate, 24)        // 3811.43

// How high can a loan be if you pay $200 for 4.5 years, at an
// annual fixed interest rate of 5%?

? PV(200, 0.05/12, 54)     // 9653.30
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:FV(), FS2:Payment(), FS2:Rate(), FS2:Periods()



# RATE ()

---

**Syntax:**

**retN = Rate ( expN1, expN2, expN3 )**

**Purpose:**

Computes the effective interest rate for a loan.

**Arguments:**

<expN1> is the loan amount.

<expN2> is the payment per installment.

<expN3> is the number of the planned payment periods.

**Returns:**

<retN> is the interest rate calculated for the payment period. 1.0 corresponds to 100%, 0.05 corresponds to 5% and so forth.

**Description:**

Rate() determines the annual interest rate for a loan <expN1> in the <expN3> period at the specified <expN2> installment. The Rate() is calculated according to Newton's iterative formula:

$$\{Sum\} \quad q = \frac{q^{n+1} - (s+1)q^n + s}{n(n+1)q^n - n(s+1)q^{n-1}} \quad \text{and} \quad s = \frac{R}{C}$$

where C = capital, R = rate per period, q = interest factor, n = number of periods

**Example:**

```
// For a $ 2500 loan, you pay $ 86.67 per month for 3 years. What
// is the effective annual interest rate?

SET DECIMALS TO 4
? Rate(2500, 86.67, 36) * 12      // 0.1501 (= 15.01% per year)
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. The CT3 error value is -1.

**Related:**

FS2:FV(), FS2:Pv(), FS2:Payment(), FS2:Periods()

# RTOD ()

---

**Syntax:**

`retN = RtoD ( expN1 )`

**Purpose:**

Converts radians to degrees.

**Arguments:**

<expN1> is a angle measurement in radians.

**Returns:**

<retN> is the result in degrees or -999.99 on error

**Description:**

With RtoD(), you can convert radians measurements into degrees.

**Example:**

```
? RtoD( PI() )           // 180.00
? RtoD(2 * PI())         // 360.00
? RtoD(4 * PI())         // 720.00
? RtoD( -PI() )          // -180.00
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:DtoR()

# SETPREC ()

---

**Syntax:**

`retN = SetPrec ( expN1 )`

**Purpose:**

Sets the precision level for trigonometric functions (CT3 only).

**Returns:**

<retN> is the currently set FS Tools precision for trigonometric functions at the time of entering the function, same as GetPrec().

**Description:**

This function is supported in FS2 for compatibility purposes to CT3 only. The setting is ignored in FS2, since the precision is same as in FlagShip: double floating point, which is on 32-bit systems in the range -  $4.94 \times 10^{-324}$  to  $1.79 \times 10^{308}$  with usually 15 significant digits. The displayed result by using the `?`, `??` or `@..SAY` etc. commands depends on the precision set by SET DECIMAL and SET FIXED. In doubt, use the `str()` function to format the numeric value with required decimal digits.

**Example:**

```
SetPrec(10)           // considered in Clipper only
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CT3 returns fix "".

**Related:**

FS2:GetPrec()

# SIGN ()

---

**Syntax:**

**retN = SIGN ( expN1 )**

**Purpose:**

Determines the mathematical sign of a number.

**Arguments:**

<expN1> is the number that is processed.

**Returns:**

<retN> is either 0 for zero value, -1 for negative, or 1 for positive value <expN1>. When <expN1> is not numeric, -999.99 signals error.

**Description:**

This function simplifies mathematical expressions, where if() constructions or functions become unnecessary.

**Example:**

```
? Sign(48335)      // 1
? Sign(-258)       // -1
? Sign(0)          // 0
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

# SIN ()

---

**Syntax:**

**retN = Sin ( expN1 )**

**Purpose:**

Computes the sine.

**Arguments:**

<expN1> is an angle in radians. If you want to specify the value in degrees, use DtoR(expN1) conversion.

**Returns:**

<retN> is the result of computation sin(expN1) or -999.99 on error

**Description:**

Sin() computes the sine value for a value in radians. The returned value is in the range of -1 to +1.

Note: the precision in FS2 is same as in FlagShip: double floating point, which is on 32-bit systems in the range - 4.94\*10<sup>-324</sup> to 1.79\*10<sup>308</sup> with usually 15 significant digits. The displayed result by using the ?, ?? or @..SAY etc. commands depends on the precision set by SET DECIMAL and SET FIXED. In doubt, use the str() function to format the numeric value with required decimal digits.

**Example:**

```
nValue := sin(0.7)
? str(nValue, 18, 15)           // 0.644217687237691

nValue := sin(0.5)
? nValue                       // 0.48
? str(nValue, 18, 15)           // 0.479425538604203

SET DECIMALS TO 10
? sin(0.5)                      // 0.4794255386
? sin(PI())                     // 0.0000000000
? sin(PI() * 99)                // 0.0000000000

? sin(DtoR(30))                 // 0.5000000000
? sin(DtoR(90))                 // 1.0000000000
? sin(DtoR(3600 + 270))         // -1.0000000000
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:Cos(), FS2:Asin(), FS2:Acos(), FS2:Tan(), FS2:Atan(), FS2:Atn2(), FS2:Cot(), FS2:DtoR(), FS2:RtoD()

# TAN ()

---

**Syntax:**

**retN = Tan ( expN1 )**

**Purpose:**

Computes the tangent.

**Arguments:**

<expN1> is an angle in radians. If you want to specify the value in degrees, use DtoR(expN1) conversion.

**Returns:**

<retN> is the result of computation tan(expN1) or -999.99 on error

**Description:**

Tan() computes the tangent value for a value in radians.

Note: the precision in FS2 is same as in FlagShip: double floating point, which is on 32-bit systems in the range - 4.94\*10<sup>-324</sup> to 1.79\*10<sup>308</sup> with usually 15 significant digits. The displayed result by using the ?, ?? or @..SAY etc. commands depends on the precision set by SET DECIMAL and SET FIXED. In doubt, use the str() function to format the numeric value with required decimal digits.

**Example:**

```
? str(Tan(PI() /9), 18, 15)    // 0.363970234266202
? str(Tan(0), 18, 15)         // 0.000000000000000
? Tan(PI() /4)                // 1.00
SET DECIMALS TO 15
? Tan(DtoR(30))               // 0.577350269189626
? Tan(DtoR(90))               // 0.000000000000000
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:Cos(), FS2:Asin(), FS2:Acos(), FS2:Sin(), FS2:Atan(), FS2:Atn2(), FS2:Cot(), FS2:DtoR(), FS2:RtoD()

# Number and Bit Manipulation

---

The FS2 Toolbox Number and Bit Manipulation provides several number and bit manipulations, e.g. conversions between two different number systems, creation of random numbers, binary operations as AND, OR, XOR, NOT, and functions to test, set and clear bits.

# BITTOC ()

---

**Syntax:**

```
retC = BitToC ( expN1, expC2, [expL3] )
```

**Purpose:**

Converts position-dependent bits into characters.

**Arguments:**

<expN1> is a number in range 0..65535 which bits should be converted to string.

<expC2> is a string with a maximum of 16 characters. Each character corresponds to a bit in <expN1>, where the last character corresponds to the lowest-value bit.

**Options:**

<expL3> is optional parameter and if .T., specifies that 0 bits are changed to blanks. The default .F. is no change. If <expL3> is .T., the resulting string length always corresponds to <expC2>.

**Returns:**

<retC> is the returned string containing the corresponding characters passed by the bit pattern.

**Description:**

The BitToC() function changes the bits of a number into a sequence of corresponding characters. This facilitates work with such bit- coded information as file attributes. Depending on the <expL3> parameter, 0 bits either displays no character (.F.) or a blank (.T.).

**Example:**

```
// The number 2 corresponds to a binary "00000010":
? BitToC(2, "ADVSHR")           // "H" as the next to last
character

// The number 5 corresponds to a binary "00000101":
? BitToC(5, "ADVSHR")           // "SR"
? BitToC(5, "ADVSHR", .T.)      // " S R"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:CtoBit(), FS2:CtoN()



# CELSIUS ()

---

**Syntax:**

**retN = Celsius ( expN1 )**

**Purpose:**

Converts a Fahrenheit temperature value into Celsius.

**Arguments:**

<expN1> is the temperature in Fahrenheit.

**Returns:**

<retN> is the converted <expN1> Fahrenheit value into Celsius (centigrade).

**Description:**

This function converts Fahrenheit temperature values directly into Celsius (centigrade) values.

**Example:**

```
? Celsius(33.8)           // 1
? Celsius(338.0)          // 170
? Celsius(3380.0)         // 1860
? Celsius(-33.8)          // -36.56
? Celsius(0)              // -17.78
? Celsius(100)            // 37.78
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:Fahrenheit()

# CLEARBIT ()

---

## Syntax:

```
retN = ClearBit ( expNC1, expN2, [expN3...expN33] )
```

## Purpose:

Clears one or more bits within a number to zero.

## Arguments:

<expNC1> is the number that is processed. Decimals, if any, will be truncated. You also may specify the number as hexadecimal string in the syntax "FF...FF" or "0xFF..FF" with max. 8 hex chars.

<expN2> ... <expN33> is the bit number to clear. Up to 32 values in any order are accepted. A valid <expN2..33> value is in the range 1..32, invalid values results in return value of -1.

## Returns:

<retN> is the result of ClearBit() or -1 on error.

## Description:

ClearBit() sets particular bit(s) to 0 within the passed number. In contrast to NumAnd(), the bit numbers are given and do not need to be converted beforehand. The <expN2..33> value 1 represents the bit with the lowest value, and the value 32 the bit with the highest value.

## Example:

```
nBitPattern := 255           // 255  bits: 0..011111111
? ClearBit(nBitPattern, 1)    // 254  = 0..011111110
? ClearBit(nBitPattern, 1, 25) // 254  = 0..011111110
? ClearBit(nBitPattern, 3)    // 251  = 0..011111011
? ClearBit(nBitPattern, 4, 3, 6) // 211 = 0..011010011

? ClearBit("FFFF", 1)        //      65535 -> 65534
? ClearBit("0xFFFFFFFF", 1, 24) // 16777215 -> 8388606
? ClearBit("xyz", 1)         // -1    (error, invalid hex)
? ClearBit(123, 35)          // -1    (error, invalid bit#)

? Num2Hex(ClearBit("FFFFFF", 1, 24)) // "7FFFFE"
? Num2Hex(ClearBit("0xFFFFFFFF", 1, 24),,.T.) // "0x7FFFFE"
? Num2Hex(ClearBit(-1, 1),,.T.) // "0xFFFFFFFFE"
? Num2Hex(ClearBit(-1, 32, 1),,.T.) // "0x7FFFFFFE"
```

## Classification:

add-on library, programming

## Compatibility:

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

## Related:

FS2:IsBit(), FS2:SetBit(), FS2:NumAnd(), FS2:NumNot(), FS2:NumOr(),  
FS2:NumXor(), Hex2num(), Num2hex(), BinAnd()

# CTOBIT ()

---

**Syntax:**

```
retN = CtoBit ( expC1, expC2 )
```

**Purpose:**

Converts a character string into a bit pattern.

**Arguments:**

<expC1> is a character sequence. When this sequence occurs in the second string, each corresponding bit is set to 1. Characters in <expC1> that are not found in <expC2> are ignored.

<expC2> is a sequence, with a maximum of 16 characters, from which the bit position is assigned

**Returns:**

<retN> is a number in the range of 0 to 65535 that corresponds to the created bit pattern

**Description:**

CtoBit() delivers a bit pattern that corresponds to a string of individual characters. When used in conjunction with its counterpart BitToC(), it facilitates work with such bit-coded information as file attributes.

**Example:**

```
? CtoBit("H", "ADVSHR")           // 2
? Num2Hex(CtoBit("H", "ADVSHR"),,.T.) // 0x02 bits: 00000010

? CtoBit("RA", "ADVSHR")           // 33
? Num2Hex(CtoBit("RA", "ADVSHR"),,.T.) // 0x21 bits: 00100001

? CtoBit("XRYA", "ADVSHR")         // 33
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:BitToC(), FS2:NtoC(), Hex2num(), Num2hex()

# CTOF ()

---

**Syntax:**

**retN = CtoF ( expC1 )**

**Purpose:**

Converts a special 8 or 9-byte string into a floating point number.

**Arguments:**

<expC1> is an 8 or 9-byte string that contains a number in the 64-bit floating point format created by FtoC() or XtoC().

**Returns:**

<retN> is the number that corresponds in the passed string, or -999.99 on error.

**Description:**

Strings created with FtoC() or XtoC() are converted back into numbers by this function. CtoF() uses different algorithm when the string was created by FtoC(num,.T.) which produces 9-byte string with additional check byte.

**Example:**

```
? CtoF(FtoC(1234.55))           // 1234.55
? CtoF(FtoC(1234.55, .T.))      // 1234.55
? CtoF(XtoC(1234.55))           // 1234.55

? CtoF(" ")                     // -999.99
? CtoF(space(8))                 // 0.0
? CtoF(space(9))                 // -999.99
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:FtoC(), FS2:XtoC()

# CTON ()

---

## Syntax:

**retN = CtoN ( expC1, [expN2], [expL3], [expL4] )**

## Purpose:

Converts a numeric string into a different base

## Arguments:

<expC1> is a string (containing numeric data) to be converted into a number in base <expN2>. Leading and trailing spaces are ignored. The data may be prefaced by positive "+" or negative "-" sign, whereby a positive sign is ignored, and negative sign interacts with <expL3>.

## Options:

<expN2> is the number base to use in the conversion. Can be in the range of 2 to 36. The default is the decimal system, base 10.

<expL3> when designated as .T., allows a negative input and/or negative result. The default allows only positive input and results. Negative input <expC1> is either prefaced by a minus "-" sign, or is determined from the first digit of <expC1> when <expN2> is 16 and when <expL3> is .T.

<expL4> is compatibility switch to CA3. When designated as .T. (the default), allows only short integers in the range 0..65535 or -32768 and +32767, same as Clipper do. Specifying .F. allows results in the range of -2147483648 to 2147483647.

## Returns:

<retN> is the converted number that corresponds to the string. On error, 0 is returned.

## Description:

CtoN() offers a number of ways to covert a number string into numeric data format. Almost any number can be converted, as long as the base <expN2> for the number system lies between 2 and 36 and the range validity succeeds.

## Example:

```
? CTON("-60000")           //      0
? CTON("60000" , , , .F.)  //    60000
? CTON("-60000" , , .T., .F.) //   -60000
? CTON("2000000")         //      0
? CTON("2000000" , , , .F.) //   2000000
? CTON("-2000000" , , .T., .F.) // -2000000
? CTON("-2000000" , , .T., .T.) //      0

? CTON("11", 2)           //      3
? CTON("1110101001100000", 2) //   60000

? CTON("A", 16)           //     10
? CTON("+ABCD", 16)       //   43981
? CTON("FFFF", 16)       //  65535
```

? CTON("FFFF", 16, .T.)	//	-1
? CTON("-FFFF", 16, .T.)	//	1
? CTON("FFFE", 16, .T.)	//	-2
? CTON("FFF", 16, .T.)	//	4095
? CTON("-FFF", 16, .T.)	//	-4095
? CTON("7FFF", 16)	//	32767
? CTON("8000", 16)	//	32768
? CTON("8000", 16, .T., .T.)	//	-32768
? CTON("8001", 16, .T., .T.)	//	-32767
? CTON("8001", 16, .T., .F.)	//	32769
? CTON("XPP", 36)	//	43981

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools, which generally supports only short integer range (0 to 65535 or -32768 to +32767), simulated in FS2 by 4th parameter set .T.

**Related:**

FS2:NtoC()

# EXPONENT ()

---

**Syntax:**

**retN = Exponent ( expN1 )**

**Purpose:**

Determines the exponent of a floating point number (base 2).

**Arguments:**

<expN1> is the number that is processed.

**Returns:**

<retN> is the exponent of the <expN1> number in base 2.

**Description:**

FlagShip floating point numbers are internally stored in IEEE double precision format. This is, simplified speaking, a byte sequence containing the logarithm (exponent) and the mantissa.

This Exponent() function determines the exponent of the <expN1> number in base 2. If the number is an integer, it will be converted to floating number first.

See also the associated function FS2:Mantissa()

**Example:**

```
// The following calculation produces the original number:
// ^2 Exponent(nValue) * Mantissa(nValue)= nValue

? Exponent(0) // Result: 0
? Exponent(Infinity()) // Result: 1023
? Exponent(100) // Result: 6

// The sign for nValue will not be considered:

? Exponent(-100) // Result: 6
? Exponent(-1.01) // Result: 0
? Exponent(-2.01) // Result: 1

// values in the range of -1 < nValue < +1, yield negative
// exponents regardless of the sign:

? Exponent(0.01) // Result: -7
? Exponent(-0.01) // Result: -7
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:Mantissa()

# FAHRENHEIT ()

---

**Syntax:**

`retN = Fahrenheit ( expN1 )`

**Purpose:**

Converts a temperature value from Celsius into Fahrenheit.

**Arguments:**

<expN1> is the temperature in Celsius (degree centigrade)

**Returns:**

<retN> is the result of the converted temperature into degrees Fahrenheit

**Description:**

This function converts temperature values from Celsius (centigrade) into Fahrenheit values.

**Example:**

```
? Fahrenheit(12.5)           // 54.50
? Fahrenheit(125.0)          // 257.00
? Fahrenheit(1250)           // 2282.00
? Fahrenheit(-155.0)         // -247.00
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:Celsius()



# FtoC ()

---

**Syntax:**

```
retC = FtoC ( expN1, [expL2] )
```

**Purpose:**

Converts a floating point number into a special 8-byte or 9-byte string.

**Arguments:**

<expN1> is the number that is processed and converted to special string.

**Options:**

<expL2> is an optional conversion mode. If specified .T., a 9-byte string is created where the 9th byte is used for validity purposes in CtoF(). If not specified or is .F., 8-byte string is created.

**Returns:**

<retC> is an 8 or 9-byte character string or "" on error.

**Description:**

FtoC() allows you to store numeric values in compressed character format. As opposite to str(), which converts numbers to string in variable size format, FtoC() stores it always in internal 8 or 9-byte format. To decode it back to number, use the CtoF() function. You may use the second parameter <expL2> for backward compatibility to FS4.48, or for very low <expN1> values, or to test the passed string for validity in CtoF().

**Example:**

```
nNum := 1234.56
? cNum := FtoC(nNum)           // chr(10,215,163,112,61,74,147,64)
? FtoC(nNum, .T.)             // chr(10,215,163,112,61,74,147,64,255)
? CtoF(cNum)                  // 1234.56
? CtoF(XtoC(nNum))            // 1234.56
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CT3 uses 8-bit string only and does not support the second parameter.

**Related:**

FS2:CtoF(), FS2:XtoC()

# INFINITY ( )

---

**Syntax:**

**retN = Infinity ( )**

**Purpose:**

Returns the largest number possible.

**Returns:**

<retN> is the largest positive floating point number.

**Description:**

Infinity() returns the largest positive floating point number.

Note: the precision in FS2 is same as in FlagShip: double floating point, which is on 32-bit systems in the range -  $4.94 \cdot 10^{-324}$  to  $1.79 \cdot 10^{308}$  with usually 15 significant digits. The displayed result by using the ?, ?? or @..SAY etc. commands depends on the precision set by SET DECIMAL and SET FIXED. In doubt, use the str() function to format the numeric value with required decimal digits.

**Example:**

```
? Infinity() // 1797693134....84124858368.00 = 1.79769313 * 10^308
? x := ltrim(str(Infinity(), 400,2)) // "1797693134...858368.00"
? len(x), at(".",x) // 312 310
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. The exact return value may slightly differ behind the 15th digit depending on the used OS.

**Related:**

FS2:Exponent(), FS2:Mantissa()

# INTNEG ()

---

**Syntax:**

```
retN = IntNeg ( expNC1, [expL2] )
```

**Purpose:**

Converts an unsigned (positive) integer into a signed integer.

**Arguments:**

<expNC1> is the number that is processed. Decimals, if any, will be truncated. You also may specify the number as hexadecimal string in the syntax "FF...FF" or "0xFF..FF" with max. 8 hex chars.

**Options:**

<expL2> if this optional parameter is specified as .T., IntNeg() processes the input value as a 32-bit unsigned integer; the default input value is a 16-bit unsigned integer (.F.).

**Returns:**

<retN> is a signed 16-bit or 32-bit integer, or 0 on error.

**Description:**

This function converts unsigned integers into signed integers. All 16-bit or 32-bit integers are accepted, depending on whether there is a second parameter and the logical value is assigned. For the 16-bit variant, all values for <expNC1> that are less than or equal to +32767 (hex=7FFF), are positive. Values in the range of +32768 (hex=8000) to +65535 (hex=FFFF) have a negative result. When you work with 32-bit values, the positive range goes to 2147483647 incl. All values beyond that are negative.

**Example:**

```
? IntNeg(0) // Result: 0 (no error)
? IntNeg(-1) // Result: 0 (error, negative val)
? IntNeg(30000) // Result: 30000
? IntNeg(32767) // Result: 32767
? IntNeg(32768) // Result: -32768
? IntNeg(32769) // Result: -32767
? IntNeg(60000) // Result: -5536
? IntNeg(65535) // Result: -1
? IntNeg(90000) // Result: 0 (error, value > 16bit)
? IntNeg("FFFF") // Result: -1
? IntNeg("7FFF") // Result: 32767
? IntNeg("D0000000", .T.) // Result: -805306368
? IntNeg("FFFFFFFF", .T.) // Result: -1
? IntNeg("FFFFFFFFFF") // Result: 0 (error, value > 16bit)
? IntNeg("GGGG") // Result: 0 (error, invalid hex)
```

**Classification:** add-on library, programming

**Compatibility:** Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:** FS2:IntPos(), FS2:NumAnd(), FS2:NumOr(), FS2:NumXor(), Hex2num()

# INTPOS ()

---

## Syntax:

**retN = IntPos ( expNC1, [expL2] )**

## Purpose:

Converts signed (plus/minus) integer into unsigned integer.

## Arguments:

<expNC1> is the number that is processed. Decimals, if any, will be truncated. You also may specify the number as hexadecimal string in the syntax "FF...FF" or "0xFF..FF" with max. 8 hex chars.

## Options:

<expL2> if this optional parameter is specified as .T., IntNeg() processes the input value as a 32-bit unsigned integer; the default input value is a 16-bit unsigned integer (.F.).

## Returns:

<retN> is an unsigned 16-bit or 32-bit integer, or 0 on error.

## Description:

This function converts signed integers into unsigned integers. All 16-bit or 32-bit integers are accepted, depending on whether there is a second parameter and the logical value is assigned. For the 16-bit variant, all values of <expNC1> that are in the range 0 to 32767 remain unchanged; values from -1 to -32768 are changed to the corresponding unsigned equivalence. When you work with 32-bit values, the positive unchanged range goes from 0 to 2147483647 and negative values from -1 to -2147483648 will be changed.

## Example:

```
? IntPos(-1)           // 65535      (largest 16-bit unsigned)
? IntPos(-2)           // 65534
? IntPos(0)             // 0
? IntPos(30000)          // 30000
? IntPos(60000)          // 60000
? IntPos(65536)          // 0          (error, value > 16bit)
? IntPos(90000)          // 24464
? IntPos(90000, .T.)     // 90000
? IntPos(-1, .T.)        // 4294967295 (largest 32-bit unsigned)
? IntPos(-90000, .T.)    // 4294877296
```

## Classification:

add-on library, programming

## Compatibility:

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

## Related:

FS2:IntPos(), FS2:NumAnd(), FS2:NumOr(), FS2:NumXor(), Hex2num()

# ISBIT ()

---

**Syntax:**

```
retL = IsBit ( expNC1, [expL2] )
```

**Purpose:**

Tests a bit in a number.

**Arguments:**

<expNC1> is the number that is processed. Decimals, if any, will be truncated. You also may specify the number as hexadecimal string in the syntax "FF...FF" or "0xFF..FF" with max. 8 hex chars.

**Options:**

<expN2> is the bit number to test. A valid range is 1 to 32. If not specified, 1 is used as default.

**Returns:**

<retL> is .T. if the designated bit is set, or is .F. otherwise

**Description:**

IsBit() tests a particular bit within a numeric field, without an requirement to mask the <expN1> using NumAnd().

**Example:**

```
nBitPattern := 253           // 253  bits: 0..011111101
? IsBit(nBitPattern, 1)      // .T.
? IsBit(nBitPattern, 2)      // .F.
? IsBit(nBitPattern, 25)     // .F.

? IsBit(-1, 1)               // .T.
? IsBit(-1, 32)              // .T.

? IsBit("0xFFFF0F", 6)      // .F.
? IsBit("0xFFFF0F", 24)     // .T.
? IsBit("0xFFFF0F", 25)     // .F.
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:SetBit(), FS2:NumAnd(), FS2:NumNot(), FS2:NumOr(), FS2:NumXor(),  
BinAnd(), BinOr(), BinXor(), BinLshift(), Num2hex(), Hex2num()

# LTON ()

---

**Syntax:**

**retN = LtoN ( [expL1] )**

**Purpose:**

Converts a logical value into a numeric value.

**Arguments:**

<expL1> is the logical value or expression to convert. If not specified or is not logical, .F. is the default.

**Returns:**

<retN> is 1 when <expL1> is .T., or is 0 otherwise

**Description:**

LtoN() converts a logical value into a numeric value. It can be used e.g. for index keys as an alternative to FS2:LtoC() function, in calculations, loops and so on.

**Example:**

```
? LtoN(.T.)           // 1
? LtoN(.F.)           // 0
? LtoN(year(date()) > 2000) // 1

? "This year has", str(365 + LtoN(IsLeap()), 3), "days."
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:CtoN(), FS2:LtoC()

# MANTISSA ()

---

**Syntax:**

**retN = Mantissa ( expN1 )**

**Purpose:**

Determines the mantissa of a floating point number (base2).

**Arguments:**

<expN1> is the number that is processed.

**Returns:**

<retN> is the mantissa of the <expN1> number. The value can be 0 or in the range of 1.0 to 2.0

**Description:**

This function supplements FS2:Exponent() to return the mantissa of the <expN1> number.

The following calculation reproduces the original value: Mantissa(expN1) \* 2 \*\* Exponent(expN1) = <expN1>

**Example:**

```
SET DECIMALS TO 6
? Mantissa(0)           // 0.000000
? Mantissa(100)         // 1.562500
? Mantissa(Infinity())  // 2.000000
? Mantissa(0.01)        // 1.280000
? Mantissa(-100)         // -1.562500
? Mantissa(-0.01)       // -1.280000
? Mantissa(-1.01)       // -1.010000
? Mantissa(-2.01)       // -1.005000
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only the first optional argument.

**Related:**

FS2:Exponent()

# NTOC ()

---

## Syntax:

```
retC = NtoC ( expNC1, [expN2], [expN3], [expC4] )
```

## Purpose:

Converts a number in a digit string of a specified number base.

## Arguments:

<expNC1> is the number that is processed. Decimals, if any, will be truncated. You also may specify the number as hexadecimal string in the syntax "FF...FF" or "0xFF..FF" with max. 8 hex chars.

## Options:

<expN2> is the number system base, from 2 to 36, to use in the result. The default is the decimal system, base 10.

<expN3> is the length of the resulting string. The maximum length is 255. The default is the length required for the conversion.

<expC4> is a pad character used to pad the resulting string on the left. The default value is a space.

## Returns:

<retC> is a string containing the conversion result or "" if the function fails, or asterisks "\*\*\*..." if the given <expN3> size is not sufficient.

## Description:

NtoC() converts a decimal number or hexadecimal string into a number string. The result is a number in base <expN2>.

## Example:

```
? NtoC (60000)                // "60000"
? NtoC (60000, 10, 8)         // " 60000"
? NtoC (60000, , 8, "$")     // "$$$60000"

? NtoC(60000, 2)              // "1110101001100000"
? NtoC("FFFF", 2)            // "1111111111111111"
? NtoC("30", 2, 8, "0")      // "00110000"

? NtoC(43981, 16)             // "ABCD"
? NtoC(43981, 36, 4)         // " XXP"

? NtoC("GGGG", 2)            // ""      (error, invalid hex)
? NtoC(60000, 10, 3)         // ""      (error, expN3 too short)
? NtoC(60000, 1, 4)         // ""      (error, invalid expN2)
```

**Classification:** add-on library, programming

**Compatibility:** Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:** FS2:CtoN(), Str(), Hex2num(), Num2hex()



# NUMAND ()

---

**Syntax:**

```
retN = NumAnd ( expNC1, expNC2, [expNC3 ...] )
```

**Purpose:**

Performs binary "AND" operation on a list of 16-bit numbers.

**Arguments:**

<expNC1>...<expNCn> are the number that are binary AND-ed. Decimals, if any, will be truncated. You also may specify the number as hexadecimal string in the syntax "FFFF" or "0xFFFF" with max. 4 hex chars. A 16-bit value is in range 0 to 65535 or -32768 to 32767.

**Returns:**

<retN> is the result of the binary AND operation or -1 on error

**Description:**

NumAnd() set the bit in the result only when both corresponding bits in <expNC1...expNCn> are set. Otherwise the resulting bit is 0.

This function makes it simple to test several bits in a number simultaneously. If bits are connected in an AND configuration, you can mask out specific bits and the unneeded bits are reset to 0.

NumAnd() uses 16-bit values only. The 32-bit counterpart is the standard function BinAnd(). To binary AND two strings, use CharAnd()

**Example:**

```
? NumAnd(0, 0) // 0
? NumAnd(0, 1) // 0
? NumAnd(1, 0) // 0
? NumAnd(1, 1) // 1
? NumAnd(1, 1, 0) // 0 : 1 AND 1 = 1 AND 0 = 0

? NumAnd(5, 7) // 5
? NumAnd(65535, "0xF0F") // 3855 = 0xF0F
? NumAnd("0xFFFF", "0xF0F") // 3855 = 0xF0F
? NumAnd("0xFFFFF", "0xFFFFF0F") // -1 (error, not 16-bit
number)
? BinAnd("0xFFFFF", "0xFFFFF0F") // 16776975 = 0xFFFFF0F

nVar := 2474 // bin: 100110101010
? "Bit 2 and 6 are " + if(NumAnd(nVar,"0x22")==0,"not ", "") + "set"
? Isbit(nVar,2), Isbit(nVar,6) // .T. .T.
```

**Classification:** add-on library, programming

**Compatibility:** Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:** FS2:NumNot(), FS2:NumOr(), FS2:NumXor(), FS2:IsBit(), FS2:SetBit(), FS2:ClearBit(), BinAnd(), Hex2num(), Num2hex()

# NUMCOUNT ()

---

**Syntax:**

**retN = NumCount ( [expN1] , [expL2] )**

**Purpose:**

Increment or set an internal counter.

**Options:**

**<expN1>** either the counters start value, or the increased counter value, in dependence on **<expL2>**. If **<expN1>** is not given or is NIL, **<retN>** returns the current counter value.

**<expL2>** if .T., the **<expN1>** is a new counter starting value, otherwise the **<expN1>** represents an counter increment. The default setting is .F.

**Returns:**

**<retN>** is the current counters value.

**Description:**

You may use NumCount() e.g. to easy number/serial different parts of your output without taking care if the real sequence number, especially if different conditional sections are used. The initial counter value is 0.

NumCount() is comparable to maintain and display a counter value, stored in a global variable.

**Example:**

```
? "output part #",ltrim( NumCount(1) )    // first part
...
? "output part #",ltrim( NumCount(1) )    // second part
...
if anything
  ? "output part #",ltrim( NumCount(1) ) // third part
  ...
endif
? "output part #",ltrim( NumCount(1) )    // third or fourth part
...
NumCount(0, .T.)                        // reset counter
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

# NUMHIGH ()

---

**Syntax:**

```
retN = NumHigh ( expNC1 )
```

**Purpose:**

Returns the high order byte of a 16-bit number.

**Arguments:**

<expNC1> is the number that is processed. Decimals, if any, will be truncated. You also may specify the number as hexadecimal string in the syntax "FF...FF" or "0xFF..FF" with max. 4 hex chars. A 16-bit value is in range 0 to 65535 or -32768 to 32767.

**Returns:**

<retN> is the high order byte of a given value, or -1 on error.

**Description:**

NumHigh() breaks down a 16-bit number into its two bytes and returns the higher order byte as a result.

**Example:**

```
nVal    := 43981
nHiVal  := NumHigh(nVal)
nLoVal  := NumLow(nVal)

? Num2hex(nVal,,.T.)           // 0xABCD
? nHiVal, "=", Num2hex(nHiVal,,.T.) // 171 = 0xAB
? nLoVal, "=", Num2hex(nLoVal,,.T.) // 205 = 0xCD

? NumHigh(80000)                // -1 (error, not 16-bit)

// To split 32-bit number into bytes, use:

nNum    := 305419896           // 0x12345678
byte1   := BinRShift(BinAnd(nNum,"0xFF000000"),24) // 18 = 0x12
byte2   := BinRShift(BinAnd(nNum,"0x00FF0000"),16) // 52 = 0x34
byte3   := BinRShift(BinAnd(nNum,"0x0000FF00"),8)  // 86 = 0x56
byte4   := BinAnd(nNum,"0xFF")                    // 120 = 0x78
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:NumLow(), BinAnd(), BinRShift(), Num2hex(), Hex2num()

# NUMLOW ()

---

**Syntax:**

**retN = NumLow ( expNC1 )**

**Purpose:**

Returns the low order byte of a 16-bit number.

**Arguments:**

<expNC1> is the number that is processed. Decimals, if any, will be truncated. You also may specify the number as hexadecimal string in the syntax "FF...FF" or "0xFF..FF" with max. 4 hex chars. A 16-bit value is in range 0 to 65535 or -32768 to 32767.

**Returns:**

<retN> is the low order byte of a given value, or -1 on error.

**Description:**

NumLow() breaks down a 16-bit number into its two bytes and returns the lower order byte as a result.

**Example:**

```
nVal := 43981
nHiVal := NumHigh(nVal)
nLoVal := NumLow(nVal)

? Num2hex(nVal,,.T.)           // 0xABCD
? nHiVal, "=", Num2hex(nHiVal,,.T.) // 171 = 0xAB
? nLoVal, "=", Num2hex(nLoVal,,.T.) // 205 = 0xCD

? NumLow(80000)                 // -1 (error, not 16-bit)

// To get low order byte of 32-bit number into bytes, use:

nNum := 305419896               // 0x12345678
byte4 := BinAnd(nNum,255)       // 120 = 0x78
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:NumHigh(), BinAnd(), Num2hex(), Hex2num()

# NUMMIRR ( )

---

**Syntax:**

**retN = NumMirr ( expNC1, [expL2] )**

**Purpose:**

Mirrors 8-bit or 16-bit values.

**Arguments:**

<expNC1> is the number that is processed. Decimals, if any, will be truncated. You also may specify the number as hexadecimal string in the syntax "FF...FF" or "0xFF..FF" with max. 4 hex chars. A 16-bit value is in range 0 to 65535 or -32768 to 32767, a 8-bit value is in range 0 to 255 or -128 to 127.

**Options:**

<expL2> specifies whether 16 bit word (.F.) or 8 bit byte (.T.) is mirrored. The default value is .F.

**Returns:**

<retN> is the mirrored byte or 16-bit number, or -1 on error.

**Description:**

NumMirr() mirrors the bits from low to height, i.e. it swaps bit 1 with bit 16 (or bit 8), bit 2 with bit 15 (or bit 7) and so forth.

**Example:**

```
? nVar := 128 + 64 + 8 + 2      // 202 = 00000000 11001010
x8 := NumMirr(nVar, .T.)
x16 := NumMirr(nVar)
? x8, NtoC(x8, 2, 16, "0")     // 83 = 00000000 01010011
? x16, NtoC(x16, 2, 16, "0")   // 21248 = 01010011 00000000

? NumMirr("0xAF5C")            // 15093 = 0x3AF5
? NumMirr("0xAF5C", .T.)       // -1 (error, not 8-bit value)
? NumMirr(1)                   // 32768
? NumMirr(1, .T.)              // 128
? NumMirr(-1)                  // 65535
? NumMirr(-1, .T.)             // 255
? NumMirr(-2)                  // 32767
? NumMirr(-2, .T.)             // 127
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:NumAnd(), FS2:NumOr(), FS2:NumHigh(), FS2:NumLow(), Num2hex(), Hex2num()

# NUMNOT ()

---

**Syntax:**

**retN = NumNot ( expNC1 )**

**Purpose:**

Performs a "NOT" operation on a 16-bit number.

**Arguments:**

<expNC1> is the number that is processed. Decimals, if any, will be truncated. You also may specify the number as hexadecimal string in the syntax "FF...FF" or "0xFF..FF" with max. 4 hex chars. A 16-bit value is in range 0 to 65535 or -32768 to 32767.

**Returns:**

<retN> is the resulting value, or -1 on error.

**Description:**

NumNot() performs binary NOT operation on a 16-bit number, i.e. will do the 1's complement: all 0 bits become 1 and 1 bits become 0.

**Example:**

```
? NumNot(0) // 65535
? NumNot(65535) // 0
? NumNot(-1) // 0
? NumNot(12345678) // -1 (error, not 16-bit value)

? NumNot("0x9625") // 38437 -> 27098
? NtoC("0x9625",2,16,"0") // 1001011000100101
? NtoC(NumNot("0x9625"),2,16,"0") // 0110100111011010
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:NumAnd(), FS2:NumOr(),FS2:NumXor(), FS2:NumHigh(), FS2:NumLow(),  
FS2:NtoC(), FS2:SetBit(), FS2:ClearBit(), Num2hex(), Hex2num()

# NUMOR ()

---

**Syntax:**

```
retN = NumOr ( expNC1, expNC2, [expNC3 ...] )
```

**Purpose:**

Performs binary "OR" operation on a list of 16-bit numbers.

**Arguments:**

<expNC1>...<expNCn> are the number that are binary OR-ed. Decimals, if any, will be truncated. You also may specify the number as hexadecimal string in the syntax "FFFF" or "0xFFFF" with max. 4 hex chars. A 16-bit value is in range 0 to 65535 or -32768 to 32767.

**Returns:**

<retN> is the result of the binary OR operation or -1 on error

**Description:**

This function joins all set bits of <expNC1...expNCn> into result value.

NumOr() uses 16-bit values only. The 32-bit counterpart is the standard FlagShip function BinOr(). To binary OR two strings, use CharOr()

**Example:**

```
? NumOr(0, 0)           // 0
? NumOr(0, 1)           // 1
? NumOr(1, 0)           // 1
? NumOr(1, 1)           // 1
? NumOr(1, 0, 1)        // 1 : 1 OR 0 = 1 OR 1 = 1

? NumOr(2, 7)           // 7
? NumOr("0x2222", "0xF0F") // 8738 -> 12079
? NumOr("0xFFFF", 1)    // 65535 -> 65535
? NumOr(255, "0x200")    // 767 = 0x2FF
? NumOr(123456, 1)       // -1 (error, not 16-bit number)
? BinOr(123456, 1)       // 123457
? BinOr("0x222222", "0x001001") // 2241059 = 0x223223
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:NumAnd(), FS2:NumNot(), FS2:NumXor(), FS2:IsBit(), FS2:SetBit(),  
FS2:ClearBit(), BinOr(), Hex2num(), Num2hex()

# NUMROL ()

---

## Syntax:

**retN = NUMROL ( expNC1, expNC2, [expN3] )**

## Purpose:

Rotate 16-bit number left or right.

## Arguments:

<expNC1> is the number that is processed. Decimals, if any, will be truncated. You also may specify the number as hexadecimal string in the syntax "FF...FF" or "0xFF..FF" with max. 4 hex chars. A 16-bit value is in range 0 to 65535 or -32768 to 32767.

<expNC1> is the number of rotation steps. The valid range is +/- 1 ... 15 (or +/- 1 ... 7 when <expN3> is .T.). When the value is positive, left rotation is performed; if negative, right rotation. Alternatively, the right rotation can be specified by value 16..30 which corresponds to -1 ... -15 (or by 8 ... 14 when <expN3> is .T.). Same as <expNC1>, you also may specify the number of rotations as hexadecimal string in the syntax "FFFF" or "0xFFFF" considering the valid range.

## Options:

<expL2> is an optional logical value. If specified .T., only the low 8-bits are rotated (low order byte), keeping the high byte untouched. The default value is .F. which rotates both the high and low order bytes together.

## Returns:

<retN> is the resulting value, or -1 on error.

## Description:

As opposite to left or right binary shift (e.g. by using the standard BinLShift() or BinRShift() FlagShip functions), which throws away the shifted-out bits and inserts 0 bits, "rotating" keeps these bits, carrying them from the highest to the lowest position (or vice versa).

## Example:

```
nVal := 1 ; nRot := NumRol(nVal, 3) // rotate bit 1 3x left
? NtoC(nVal, 2, 16, "0"), ">";
  NtoC(nRot, 2, 16, "0") // 0000000000000001 -> 0000000000001000

? NumRol(nVal,-3)           // rotate bit 1 3x right (same as ..,13)
                           // 0000000000000001 -> 0010000000000000

? NumRol(33282, 3)          // rotate bits 4,10,16 3x left
                           // 1000001000000010 -> 0001000000010100
? NumRol(33282,-3)          // rotate bits 4,10,16 3x right
                           // 1000001000000010 -> 0101000001000000

? NumRol(26, 2, .T.)        // rotate bits 2,4,5 2x left, lower byte
                           // 00011010 -> 01101000
? NumRol(26, -2, .T.)       // rotate bits 2,4,5 2x right, lower byte
```



```

// 00011010 -> 10000110
? NumRot(60000,2,.T.) // rotate bit 6,7 in lower byte 2x left
// 11101010 01100000 -> 11101010 10000001
? NumRot(11234567, 2) // -1 (error, invalid parameter value)
? NumRot(1, "0xFF") // -1 (error, invalid parameter value)

```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:FS2:NumHigh(), FS2:NumLow(), FS2:NtoC(), BinLShift(), BinRShift(),  
Num2hex(), Hex2num()

# NUMXOR ()

---

**Syntax:**

**retN = NumXor ( expNC1, expNC2, [expNC3 ...] )**

**Purpose:**

Performs binary exclusive "OR" operation on a list of 16-bit numbers.

**Arguments:**

<expNC1>...<expNCn> are the number that are binary XOR-ed. Decimals, if any, will be truncated. You also may specify the number as hexadecimal string in the syntax "FFFF" or "0xFFFF" with max. 4 hex chars. A 16-bit value is in range 0 to 65535 or -32768 to 32767.

**Returns:**

<retN> is the result of of the binary XOR operation or -1 on error

**Description:**

This function set the bit in the result when both corresponding bits in <expNC1...expNCn> differs. Otherwise the resulting bit is 0. The X-OR operation is reversible, when the same mask is used.

NumXor() uses 16-bit values only. The 32-bit counterpart is the standard FlagShip function BinXor(). To binary XOR two strings, use CharXor()

**Example:**

```
? NumXor(0, 0) // 0
? NumXor(0, 1) // 1
? NumXor(1, 0) // 1
? NumXor(1, 1) // 0
? NumXor(1, 0, 1) // 0 : 1 XOR 0 = 1 XOR 1 = 0

? res := NumXor("0x2222", "0x101") // 8995 = 0x2323
? NumXor(res, "0x101") // 8738 = 0x2222
? NumXor(res, "0x2222") // 257 = 0x101

? NumXor(123456, 1) // -1 (error, not 16-bit number)
? BinXor(123456, 1) // 123457
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:NumAnd(), FS2:NumNot(), FS2:NumOr(), FS2:IsBit(), FS2:SetBit(),  
FS2:ClearBit(), BinXor(), Hex2num(), Num2hex()

# RAND ()

---

**Syntax:**

**retN = Rand ( [expN1] )**

**Purpose:**

Generates pseudo-random numbers.

**Options:**

<expN1> is the start value for the random number generator. Only the integer part of <expN1> is considered. If not yet specified, or when 0 (zero) is given, Rand() initializes the generator with the current Milliseconds time, i.e. with Seconds() \* 1000.

**Returns:**

<retN> is non-negative floating-point values between 0.0 and 1.0

**Description:**

Rand() creates pseudo-random numbers using the linear congruential algorithm. When Rand() is called without parameter, every call returns different value compared to the previous call.

The algorithm depends heavy on the init value. In its dependence, the subsequent return values are calculated from the linear series which means: initializing Rand() with the same value will create the same random number sequence. Therefore, use Random(0) or other random values for the initialization instead of constants, see example below.

**Example:**

```
SET DECIMALS TO 15
? Rand()           // 0.417117540798248   (auto-initialized)
? Rand()           // 0.323154410670639
? Rand()           // 0.414932909692777

? Rand(23)         // 0.199281135414825   (a)
? Rand()           // 0.955482658089267   (b)
? Rand()           // 0.080619594120559   (c)
? Rand(23)         // 0.199281135414825   (same as a)
? Rand()           // 0.955482658089267   (same as b)
? Rand()           // 0.080619594120559   (same as c)

? Rand(0)          // 0.316372948772372   (d)
? Rand()           // 0.233325552518608   (e)
? Rand()           // 0.137626410733116   (f)
? Rand(0)          // 0.670384492100315   (differs from d)
? Rand()           // 0.756277873736934   (differs from e)
? Rand()           // 0.829854223449853   (differs from f)
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CT3 uses a fix 100001 init value, instead of a random value. FS2 internally use system functions `drand48()` and `srand48()`

**Related:**

FS2:Random()

# RANDOM ()

---

**Syntax:**

**retN = Random ( [expLN1] , [expN2] )**

**Purpose:**

Generates random numbers.

**Options:**

<expLN1> specifies the Random() precision and CT3 compatibility.

- if <expLN1> is not given or is NIL, the default random numbers range is between 0 and 2147483647.
- if specified .F., generates random numbers between 0 and 65535
- if specified .T., generates random numbers between -32768 to +32767
- if <expLN1> is numeric in range 1 to 65535, Random() generates only numbers in the range 0 to the given <expLN1> number.

<expN2> is the start value for the random number generator. Only the integer part of <expN1> is considered. If not yet specified, or when 0 (zero) is given, Random() initializes the generator with the current Milliseconds time, i.e. with Seconds() \* 1000.

**Returns:**

<retN> is an integer random number, in the range 0 to 2147483647 (or 0 to 65535 or -32768 to +32767 or 0..n depending on <expLN1>).

**Description:**

Random() creates random numbers using non-linear additive feedback random number generator to return successive pseudo-random numbers in the range from 0 to 2147483647 or in the specified range. The period of this random number generator is very large, approximately  $16^{((2^{31})-1)}$

Using the numeric or logical <expLN1> value generates wider range of random numbers than using modulo, i.e. Random(16) is usually better than Random() % 16.

**Example:**

```
for ii := 1 to 10
  ? Random(), Random(.F.), Random(.T.), Random(10)
next

// 1935990818      61380      2033      3
// 1733384307      27404      -5146     6
// 275801208       46066     -18640     4
// 419823          47801      336      3
// 773134563       4723     -32340     0
// 1919866410      17145     -30815     8
// 39885523        53748      2145      5
// 577516522       2194      22893      1
```

// 2083272147	34926	-32670	7
// 1714792753	27721	27202	9

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CT3 supports only the first logical argument.

**Related:**

FS2:Rand()

# SETBIT ()

---

**Syntax:**

```
retN = SetBit ( expNC1, expN2, [expN3...expN33] )
```

**Purpose:**

Sets one or more bits within a number.

**Arguments:**

<expNC1> is the number that is processed. Decimals, if any, will be truncated. You also may specify the number as hexadecimal string in the syntax "FF...FF" or "0xFF..FF" with max. 8 hex chars.

<expN2> ... <expN33> is the bit number to set. Up to 32 values in any order are accepted. A valid <expN2..33> value is in the range 1..32, invalid values results in return value of -1.

**Returns:**

<retN> is the result of SetBit() or -1 on error.

**Description:**

SetBit() sets particular bit(s) to 1 within the passed number. In contrast to NumOr() or BinOr(), the bit numbers are given and do not need to be converted beforehand. The <expN2..33> value 1 represents the bit with the lowest value, and the value 32 the bit with the highest value.

**Example:**

```
nBitPattern := 96 // 96 bits: 0..001100000
? SetBit(nBitPattern, 1) // 97 = 0..001100001
? SetBit(nBitPattern, 1, 4) // 105 = 0..001101001

? SetBit(nBitPattern, 3, 4, 7, 9) // 364 = 0..101101100
? NumOr (nBitPattern, "0x14C") // 364 = 0..101101100

? SetBit("0xFFFF", 1) // 65520 -> 65521
? SetBit("0xFFFFA", 1, 25) // 65520 -> 16842747
? SetBit(0, 32) // 0 -> -2147483648

? SetBit("xyz", 1) // -1 (error, invalid hex)
? SetBit(123, 35) // -1 (error, invalid
bit#)
? SetBit(-4, 1) // -3
? SetBit(250.2, 1) // 251
```

**Classification:** add-on library, programming

**Compatibility:** Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:** FS2:IsBit(), FS2:ClearBit(), FS2:NumAnd(), FS2:NumNot(), FS2:NumOr(), FS2:NumXor(), Hex2num(), Num2hex(), BinOr()

# Date, Time Functions & Triggers

---

## Introduction

With the FS2 Toolbox **Date and Time** functions, you can manipulate and calculate date and time, used often for certain technical and financial computations.

## Index of FS2 Date & Time Functions

ADDMONTH()	Adds or subtracts months to/from a date
BOM()	Determines the date of the first day of a month
BOQ()	Determines the date for the beginning of a quarter
BOY()	Determines the date for the beginning of a year
CDOW2()	Converts date value to a character weekday
CMONTH2()	Converts date value to a character month
CTODOW()	Converts the day of the week name into a corresponding number
CTOMONTH()	Converts the name of the month into a corresponding number
DMY()	Returns a date in "DD Month YY" format
DOY()	Determines the day of the year for a specific date
EOM()	Determines the date for the last day of a month
EOQ()	Determines the date for the end of a quarter
EOY()	Determines the date for the end of the year
ISLEAP()	Tests if a specific year is a leap year
KEYSEC()	Triggers a key trap after a time delay
KEYTIME()	Triggers a key trap at a specific clock time
LASTDAYOM()	Determines the number of days in a month
MDY()	Returns a date in the "Month DD, YY" format
NTOCDOW()	Converts the number of a weekday into a weekday name
NTOCMONTH()	Converts the number of a month into a month name
QUARTER()	Determines the quarter in which a specific date lies
SECTOTIME()	Converts seconds into a time string
SETDATE()	Sets the system date
SETTIME()	Sets the system clock
SHOWTIME()	Continuously displays the time at desired screen position
TIMETOSEC()	Convert the character time value to seconds since midnight
TIMEVALID()	Determines whether a specified time is valid
TRIGGERUDF()	Triggers specified UDF or code block after a time delay
WAITPERIOD()	Checks if specified time period is expired
WEEK()	Returns the calendar week for a date
WEEKSETISO()	Returns or set ISO compatibility for WEEK()
WOM()	Returns the week of a month



# ADDMONTH ()

---

**Syntax:**

**retD = AddMonth ( [expD1], expN2 )**

**Purpose:**

Adds or subtracts months to/from a date.

**Arguments:**

<expD1> is the date value that is processed. If not specified or is NIL, the current date() value is used.

<expN2> is the number of months to add to <expD1> or subtract from.

Note: If the <expD1> is numeric and only one parameter is available, a parameter shift is assumed and <expD1> is treated as <expN2>. Such a parameter shift is a bad programming technique and not suggested, but handled by FS2 for compatibility purposes to CT3.

**Returns:**

<retD> is the new date value or empty date on error.

**Description:**

AddMonth() permits you to add months to a given or current date. You may use it to calculate payment due dates based on an invoice date and for similar purposes. If you use a negative <expN2> number, months are subtracted. If the new day value exceeds the number of days in this month, the last month's day (28/29/30/31) is used.

**Example:**

```
? date() // 03/22/02
? AddMonth( date(), 1) // 04/22/02
? AddMonth( , 1) // 04/22/02
? AddMonth(1) // 04/22/02
? AddMonth() // / / (= error)

dMaxFeb := ctod("02/29/2004")
SET CENTURY ON
? dMaxFeb, dMaxFeb + 365 // 02/29/2004 02/28/2005
? AddMonth( dMaxFeb, 12) // 02/28/2005
SET DATE ANSI
? dMaxFeb // 2004.02.29
? AddMonth( dMaxFeb, 12) // 2005.02.28
? AddMonth( dMaxFeb, 48) // 2008.02.29
? AddMonth( dMaxFeb, -96) // 1996.02.29

? "This invoice is due on", AddMonth(, 2) // = in two months
? "This invoice is due on", date() + 60 // = in 60 days
```

**Classification:** add-on library, programming

**Compatibility:** Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:** Date(), Month(), Day(), Year(), Dtoc(), Dtos(), Ctod(), Stod(), Empty(), FS2:IsLeap(), FS2:EoM(), FS2>LastDayOm()

# BOM ()

---

**Syntax:**

**retD = BoM ( [expD1] )**

**Purpose:**

Determines the date of the first day of a month for given date.

**Options:**

<expD1> is a date for which the first day of the month is determined. If not specified, the system date() is used.

**Returns:**

<retD> is the date value containing the first day of the month in which <expD1> lies, or empty date on error.

**Description:**

BoM() allows you to determine the first day of the month that contains <expD1>. This may be used e.g. to determine how many days have past since the beginning of the month, see example below.

**Example:**

```
? date()           // 03/22/02
? BoM()            // 03/01/02
? date() - BoM()    // 21

? BoM(ctod("11/10/02")) // 11/01/02
? BoM(ctod("13/10/02")) // / / (= error)
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:EoM(), FS2:BoQ(), FS2:EoQ(), FS2:BoY(), FS2:EoY(), Date(), Month(), Day(), Year(), Dtoc(), Dtos(), Ctod(), Stod()

# BOQ ()

---

**Syntax:**

`retD = BoQ ( [expD1] )`

**Purpose:**

Determines the date of the first day of a quarter for given date.

**Options:**

<expD1> is a date for which the first day of the quarter is determined. If not specified, the system date() is used.

**Returns:**

<retD> is the date value containing the first day of the quarter in which <expD1> lies, or empty date on error.

**Description:**

BoQ() allows you to determine the first day of the quarter that contains <expD1>. This may be used e.g. to determine how many days have past since the beginning of the quarter, see example below.

**Example:**

```
? date()           // 03/22/02
? BoQ()            // 01/01/02
? date() - BoQ()    // 80

? BoQ(ctod("11/10/02")) // 10/01/02
? BoQ(ctod("13/10/02")) //  /  /  (= error)
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:EoQ(), FS2:BoM(), FS2:EoM(), FS2:BoY(), FS2:EoY(), Date(), Month(), Day(), Year(), Dtoc(), Dtos(), Ctod(), Stod()

# BOY ()

---

**Syntax:**

**retD = BoY ( [expD1] )**

**Purpose:**

Determines the date of the first day of a year for given date.

**Options:**

<expD1> is a date for which the first day of the year is determined. If not specified, the system date() is used.

**Returns:**

<retD> is the date value containing the first day of the year in which <expD1> lies, or empty date on error.

**Description:**

BoM() allows you to determine the first day of the year that contains <expD1>. This may be used e.g. to determine how many days have past since the beginning of the year, see example below.

**Example:**

```
SET CENTURY ON
? date()           // 03/22/2002
? BoY()            // 01/01/2002
? date() - BoY()   // 80

? dDate := ctod("11/10/02") // 11/10/2002
? BoY(dDate)       // 01/01/2002
? dDate - BoY(dDate) // 313

? BoY(ctod("13/10/02")) // / / (= error)
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:EoY(), FS2:BoM(), FS2:EoM(), FS2:BoQ(), FS2:EoQ(), Date(), Month(), Day(), Year(), Dtoc(), Dtos(), Ctod(), Stod()

# CDOW2 ( )

---

**Syntax:**

**retC** = Cdw2 ( [expD1] )

**Purpose:**

Converts date value to a character weekday.

**Arguments:**

<expD1> is the date value. If not specified, the current system date() is used.

**Returns:**

<retC> is a character string, containing the name of the day of the week (e.g. "Monday" or "Montag").

**Description:**

Cdw2() is comparable to the standard FlagShip function Cdw(), but Cdw2() is independent on the current language setting via FS\_SET("setlang"|loadlang"), using the day-of-week definitions in the fs2\_udfs.prg source.

Unlike the similar NtoCdw() which accepts numeric value of the weekday number, this Cdw2() determines the day from the passed date value.

For a full internationalization, this function is available in source code in the fs2\_udfs.prg. The default language is English, but you can modify the day names (in the array at the beginning of the source code) to fit to your native language.

**Example:**

```
? Cdw2()                // "Monday"
? Cdw2(date() + 2)       // "wednesday"
? Cdw2()                 // "anything-else" (if changed)

? Cdw()                  // "Monday"
? Cdw(date() + 3)        // "Thursday"
? Cdw(ctod("08/20/02"))  // "Tuesday"

FS_SET ("load", 1, "FSSortab.ger")
FS_SET ("set1", 1)
? Cdw()                  // "Montag"
? Cdw2()                 // "Monday"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, not available in CA3tools.

**Related:**

FS2:NtoCdw(), FS2:Cmonth2(), FS2:CtoMonth(), FS2:NtoCmonth(), Cdw(), Dow(), Date(), Month(), Day(), Year(), Dtoc(), Dtos()

# CMONTH2 ()

---

**Syntax:**

```
retC = Cmonth2 ( [expD1] )
```

**Purpose:**

Converts date value to a character month.

**Arguments:**

<expD1> is the date value. If not specified, the current system date() is used.

**Returns:**

<retC> is a character string, containing the name of the month (e.g. "January" or "Januar").

**Description:**

Cmonth2() is comparable to the standard FlagShip function Cmonth(), but Cmonth2() is independent on the current language setting via FS\_SET("setlang"|"loadlang"), using the month definitions in the fs2\_udfs.prg source.

Unlike the similar NtoMonth() which accepts numeric value of the weekday number, this Cmonth2() determines the month from the passed date value.

For a full internationalization, this function is available in source code in the fs2\_udfs.prg. The default language is English, but you can modify the day names (in the array at the beginning of the source code) to fit to your native language.

**Example:**

```
? Cmonth2()           // "March"
? Cmonth2(date() + 60) // "May"
? Cmonth2(ctod("08/20/02")) // "August"
? Cmonth2()           // "anything-else" (if changed)

? Cmonth()           // "March"
? Cmonth( AddMonth(2) ) // "May"
? Cmonth(ctod("08/20/02")) // "August"

FS_SET("load", 1, "FSsortab.ger")
FS_SET("set1", 1)
? Cmonth()           // "Maerz"
? Cmonth2()          // "March"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, not available in CA3tools.

**Related:**

FS2:NtoCmonth(), FS2:Cdow2(), FS2:CtoMonth(), FS2:NtoCdow(), Cmonth(), Dow(), Date(), Month(), Day(), Year(), Dtoc(), Dtos()

# CTODOW ()

---

**Syntax:**

**retN** = CtoDow ( **expC1** )

**Purpose:**

Converts the name of the day of the week into a corresponding number.

**Arguments:**

<**expC1**> is a string that contains the name of a day of the week. The given name is case insensitive. You can abbreviate the day names, but these abbreviations must be explicit to return an accurate value.

**Returns:**

<**retN**> is a number 1 to 7 corresponding to the given day name, or 0 if the function fails.

**Description:**

You may use CtoDow() to change the name of a day of the week to a number. Sunday corresponds to 1, Monday 2, ... and Saturday 7.

For a full internationalization, this function is available in source code in the fs2\_udfs.prg. The default language is English, but you can modify the day names (in the array at the beginning of the source code) to fit to your native language.

**Example:**

```
? CtoDow("Sunday")           // 1
? CtoDow("WEDNESDAY")        // 4
? CtoDow("wednesday")        // 4
? CtoDow(" wed ")            // 4

? CtoDow("m")                // 2 = found: Monday
? CtoDow("sa")                // 7 = found: Saturday
? CtoDow("s")                 // 1 = found: Sunday
? CtoDow("P")                 // 0 = not found

? NtoCdw(CtoDow("t"))         // "Tuesday"
? NtoCdw(CtoDow("th"))        // "Thursday"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools does not support modifiable internationalization.

**Related:**

FS2:NtoCdw(), FS2:CtoMonth(), FS2:NtoCmonth(), FS2:Cdw2(), Dow(), Date(), Month(), Day(), Year(), Dtoc(), Dtos(), Ctod()

# CTOMONTH ()

---

**Syntax:**

**retN = CtoMonth ( expC1 )**

**Purpose:**

Converts the name of the month into a corresponding number.

**Arguments:**

<expC1> is a string that contains the name of the month. The given name is case insensitive. You can abbreviate the months, but these abbreviations must be explicit to return an accurate value.

**Returns:**

<retN> is a number 1 to 12 corresponding to the given month, or 0 if the function fails.

**Description:**

You may use CtoMonth () to change the name of a month to a number. January corresponds to 1, February 2, ... and December 12

For a full internationalization, this function is available in source code in the fs2\_udfs.prg. The default language is English, but you can modify the months (in the array at the beginning of the source code) to fit to your native language.

**Example:**

```
? CtoMonth("January")           // 1
? CtoMonth("AUGUST")            // 8
? CtoMonth(" August ")         // 8

? CtoMonth("a")                 // 4 = found: April
? CtoMonth("au")                // 8 = found: August
? CtoMonth("p")                 // 0 = not found

? NtoCmonth(CtoMonth("s"))      // "September"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools does not support modifiable internationalization.

**Related:**

FS2:NtoCmonth(), FS2:CtoDow(), FS2:NtoCdown(), FS2:Cmonth2(), Date(), Month(), Day(), Year(), Dtoc(), Dtos(), Ctod()



# DMY ()

---

## Syntax:

**retC** = **DMY** ( [**expD1**] , [**expL2**] , [**expL3**] )

## Purpose:

Returns a date in "DD Month [YY]YY" format.

## Options:

<**expD1**> is the date value that is processed. If not specified, the system date() is used.

<**expL2**> is a logical value specifying whether to insert a period (.) after the day. The default is no period (.F.)

<**expL3**> is a logical value: .T. specifies to use month names defined in FSsortab.\* and assigned by standard FlagShip function Cmonth(); .F. or NIL specifies to use month names defined in the source file fs2\_udfs.prg same as Cmonth2()

Note: If the <expD1> is logical, a parameter shift is assumed and <expD1> is treated as <expL2>. Such a parameter shift is a bad programming technique and not suggested, but handled by FS2 for compatibility purposes to CT3.

## Returns:

<**retC**> is the resulting string in "DD[.] Month [YY]YY" format, or "" if the function fails.

## Description:

Use DMY() to return the date as a string using the day, name of month, and year. The SET CENTURY ON/OFF switch determines whether or not the year is displayed in two or four digits.

If the <expL2> parameter is designated .T., the function builds in a "." after the day designation.

When the <expL3> is set .T., the function uses internationalization specified by FS\_SET("setlang"|"loadlang") and the month names are assigned from the file assigned by FS\_SET("setlang", file).

For a full internationalization, this function is available in source code in the fs2\_udfs.prg. The default language is English, but you can modify the months (in the array at the beginning of the source code) to fit to your native language.

## Example:

```
SET CENTURY OFF
? DMY()                // "7 March 03"
? DMY(.T.)             // "7. March 03"

FS_SET ("load", 1, "FSsortab.ger")
FS_SET ("set1", 1)

SET CENTURY ON
```

```
? DMY(ctod("02/01/89"), .T.)           // "1. February 1989"  
? Cdw2() + ", " + DMY(.T.)             // "Friday, 7. March 2003"  
  
? DMY(.T., .T.)                       // "7. Maerz 2003"  
? Cdw() + ", " + DMY(.T.,.T.)         // "Freitag, 7. Maerz 2003"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools support only the two first arguments and does not support modifiable internationalization.

**Related:**

FS2:MDY(), FS2:NtoCmonth(), FS2:NtoCdw(), FS2:Cmonth2(), Date(),  
WebDate(), Month(), Cmonth(), Day(), Year(), Dtoc(), Dtos(), Ctod(), Stod()

# DOY ()

---

**Syntax:**

**retN = DoY ( [expD1] )**

**Purpose:**

Determines the calendar day number for a specific date.

**Options:**

<expD1> is the date value that is processed. If not specified, the system date() is used.

**Returns:**

<retN> is a calendar day number that specifies which day of the year the <expD1> represents, or 0 if the function fails

**Description:**

DoY() returns a calendar day number to a date that relates to the beginning of the year. This function is useful for calculating of date differences or time periods. However, you also may use addition and subtraction of date values as well.

**Example:**

```
? "Today is the", ltrim(DoY()) + ". calendar day in", year()

dxmas := ctod("12/24/" + ltrim(year()) )
? "Christmas is the", ltrim( DoY(dxmas) ) + ". calendar day " + ;
  "of total", ltrim( DoY(EoY()) ), "days this year"

? "Yet", ltrim(str(dxmas - date(),3,0)), "days until Christmas"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:BoY(), FS2:EoY(), FS2:Quarter(), FS2:LastDayOm(), Date(), Month(), Day(), Dow(), Year(), Dtoc(), Dtos(), Ctod()

# EOM ()

---

**Syntax:**

**retD = EoM ( [expD1] )**

**Purpose:**

Determines the date of the last day of a month for given date.

**Options:**

<expD1> is a date for which the last day of the month is determined. If not specified, the system date() is used.

**Returns:**

<retD> is the date value containing the last day of the month in which <expD1> lies, or empty date on error.

**Description:**

EoM() allows you to determine the last day of the month that contains <expD1>. This may be used e.g. in invoices due at the month end, or in calculations how many days remain until the end of the month, etc.

Unlike LastDayOm(), this EoM() does not return a numeric value (the number of days in the month), but a date value which can be used in addition or subtraction with other dates.

**Example:**

```
? date()           // 03/22/02
? BoM()            // 03/01/02
? EoM()            // 03/31/02
? date() - BoM()    // 21
? EoM() - date()    // 9

? BoM(ctod("11/10/02")) // 11/01/02
? EoM(ctod("11/10/02")) // 11/30/02
? EoM(ctod("13/10/02")) // / / (= error)
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:BoM(), FS2:BoQ(), FS2:EoQ(), FS2:BoY(), FS2:EoY(), FS2>LastDayOm(), Date(), Month(), Day(), Year(), Dtoc(), Dtos(), Ctod(), Stod()

# EOQ ()

---

**Syntax:**

`retD = EoQ ( [expD1] )`

**Purpose:**

Determines the date of the last day of a quarter for given date.

**Options:**

<expD1> is a date for which the last day of the quarter is determined. If not specified, the system date() is used.

**Returns:**

<retD> is the date value containing the last day of the quarter in which <expD1> lies, or empty date on error.

**Description:**

EoQ() allows you to determine the last day of the quarter that contains <expD1>. This may be used e.g. in calculations how many days remain until the end of the quarter.

**Example:**

```
SET CENTURY ON
? date()                // 02/22/2002
? BoQ()                 // 02/01/2002
? EoQ()                 // 03/31/2002
? date() - BoQ()        // 37
? EoQ() - date()        // 52

? BoQ(ctod("11/10/02")) // 10/01/2002
? EoQ(ctod("11/10/02")) // 12/31/2002
? EoQ(ctod("13/10/02")) //   /   /   (= error)
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:BoQ(), FS2:BoM(), FS2:EoM(), FS2:BoY(), FS2:EoY(), Date(), Month(), Day(), Year(), Dtoc(), Dtos(), Ctod(), Stod()

# EOY ()

---

**Syntax:**

**retD = EoY ( [expD1] )**

**Purpose:**

Determines the date of the last day of a year for given date.

**Options:**

<expD1> is a date for which the last day of the year is determined. If not specified, the system date() is used.

**Returns:**

<retD> is the date value containing the last day of the year in which <expD1> lies, or empty date on error. The last day is always December 31 of the specified year.

**Description:**

EoM() allows you to determine the last day of the year that contains <expD1>. This may be used e.g. in calculations how many days remain until the end of the year.

**Example:**

```
SET CENTURY ON
? date()           // 03/22/2002
? BoY()            // 01/01/2002
? EoY()            // 12/31/2002
? date() - BoY()   // 80
? EoY() - date()   // 284

SET CENTURY OFF
? dDate := ctod("11/10/02") // 11/10/02
? BoY(dDate)         // 01/01/02
? EoY(dDate)         // 12/31/02
? dDate - BoY(dDate) // 313
? EoY(dDate) - dDate // 51

? EoY(ctod("13/10/02")) // / / (= error)
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:BoY(), FS2:BoM(), FS2:EoM(), FS2:BoQ(), FS2:EoQ(), Date(), Month(), Day(), Year(), Dtoc(), Dtos(), Ctod(), Stod()

# ISLEAP ()

---

**Syntax:**

**retL = IsLeap ( [expD1] )**

**Purpose:**

Tests if a specific year is a leap year.

**Options:**

<expD1> is the date value that is processed. If not specified, the system date() is used.

**Returns:**

<retL> is .T. when <expD1> lies in a leap year, otherwise .F. is returned.

**Description:**

A number of technical financial calculations need to know if a year has 365 or 366 days or if February has 28 or 29 days. IsLeap() determines this.

**Example:**

```
? date() // 03/22/2002
? IsLeap() // .F.
? IsLeap(ctod("01/01/1900")) // .F.
? IsLeap(ctod("09/30/1996")) // .T.
? IsLeap(ctod("01/01/2000")) // .T.
? IsLeap(ctod("05/07/2004")) // .T.

? IsLeap(ctod("13/32/2004")) // .F. (error)
? DateValid(ctod("13/32/2004")) // .F.

? EoM(ctod("02/01/" + ltrim(year()))) // 02/28/02
? EoM(ctod("02/01/2000")) // 02/29/00
? EoM(ctod("02/01/2004")) // 02/29/04
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools which support only syntax 1 or no parameters at all to disable.

**Related:**

FS2:EoM(), Date(), DateValid(), Month(), Day(), Year(), Dtoc(), Dtos(), Ctod(), Stod()

# KEYSEC ()

---

## **Syntax1 :**

```
retL = KeySec ( expN1, expN2, [expN3], [expL4],  
               [expL5] )
```

## **Syntax 2:**

```
retL = KeySec ( [expL1] )
```

## **Purpose:**

Triggers a key trap after a time delay.

## **Arguments:**

<expL1> in syntax 2 is an optional logical parameter. If specified .T., the KeySec() function is triggered enforced, i.e. independent of the current time value. If specified .F. or no parameters are given, KeySec() is uninstalled.

<expN1> in syntax 1 is numeric value specifying the Inkey() code to place into the keyboard buffer. When <expN1> is 0 (zero), KeySec(0) return the current trigger status: .T. if the trigger is active and .F. otherwise. In this case, <expN2> can be omitted.

<expN2> is numeric value specifying the time interval in seconds after which KeySec() is triggered. The valid range is 1 to 86400 (1sec to 24hr). For CT3 compatibility, negative values are supported too and specify multiples of 1/18.2 seconds; values less than 1 second are rounded to 1 sec.

<expN3> is a numeric value specifying the count = number of times the function is triggered before it uninstalls itself. The default value is 1, which uninstall KeySec() after the first invocation. A value of -1 designates to trigger KeySec() forever until it is manually uninstalled by KeySec(.F.) or the program ends.

<expL4> is a logical value specifying that the <expN3> counter should re-start after the <expN1> key is pressed or the key was removed from keyboard buffer by Inkey\*(), KEYBOARD w/o ADDITIVE clause, CLEAR TYPEAHEAD or by SET TYPEAHEAD. The default .F. value does not check the keyboard buffer and hence does not restart the counter once it expires.

<expL5> is a logical value specifying that the <expN1> key value should be added into the keyboard buffer, same as ADDITIVE clause of the KEYBOARD command. The default .F. value specifies that the <expN1> value is placed unconditionally and overwrites all type- ahead values in the keyboard buffer.

## **Returns:**

<retL> is .T. on success, or .F. if the KeySec() installation or de-installation fails.

## **Description:**

This function is useful to terminate READ after desired time, or in demo programs to simulate a keyboard input with a time delay. It also allows to automatically place data into the keyboard buffer.



In conjunction with SET KEY or ON KEY, you may perform any user action once the <expN1> key is removed from the buffer by Inkey(), InkeyTrap(), WAIT or READ. Note this difference to the TriggerUdf() function which is invoked directly, without the detour via keyboard buffer and hence without requirement of Inkey\*() or wait state.

The specified <expN1> value is placed into the keyboard buffer every time after the time interval <expN2> expires, starting at installation of KeySec() or after the manual invocation via KeySec(.T.). You will need to invoke KeyTime(.T.) from time to time in UDFs written in C, where the automatic trigger is inactive. When the number of invocations <expN3> expires, KeySec() de-installs itself except it is prevented by negative <expN3> or by setting of <expL4>. You may de-install it manually by invoking KeySec() without parameter or by KeySec(.F.).

Note that the use of KeySec() may slow-down the execution speed. KeySec() may coexist with other triggers like ShowTime(), KeyTime(), and TriggerUdf().

Also note, that in the multi-user and multi-tasking environment, there is no guarantee for an exact time triggering, since other applications are executed parallel (in time slices) which may have higher priority (like system or kernel tasks) and may receive larger time slice. Therefore the exact definition of KeySec() is "execute at or after the given time value expires".

As opposite to Clipper/CT3, you don't need to de-activate KeySec() before the program ends, FS2 do it automatically.

**Example:**

```
local cVar := space(20)
SET KEY K_CTRL_F20 TO myAbortRead // invoke this UDF on Ctrl-F20
if !KeySec(K_CTRL_F20, 10)        // start KeySec(), pass Ctrl-F20
    alert("could not install key trapping")
endif
@ 1,1 SAY "enter data within 10 seconds" GET cVar
READ

KeySec()                          // uninstall KeySec()
SET KEY K_CTRL_F20 TO            // uninstall SET KEY

@ 3,1 say "read value = '" + cVar + "'"
wait "done..."

FUNCTION myAbortRead(p1,p2,p3)
local oGet := GetActive()
if valtype(oGet) == "o" .and. !empty(oGet)
    oGet:VarPut( padr("aborted", len(oGet:Buffer) ) )
    CLEAR GETS
endif
return
```

**Classification:** add-on library, programming

**Compatibility:** Available in FS2-Toolbox, compatible to NT2/CA3 tools which support only syntax 1 with max four params, or no parameters at all.

**Related:** FS2:TriggerUdf(), FS2:KeyTime(), FS2:ShowTime(), Seconds(), Inkey(), InkeyTrap(), KEYBOARD

# KEYTIME ()

---

## **Syntax1 :**

**retL = KeyTime ( expN1, expC2, [expL3] )**

## **Syntax 2:**

**retL = KeyTime ( [expL1] )**

## **Purpose:**

Triggers a key trap at a specific clock time.

## **Arguments:**

**<expL1>** in syntax 2 is an optional logical parameter. If specified .T., the KeyTime() function is triggered enforced, i.e. independent of the current time value. If specified .F. or no parameters are given, KeyTime() is uninstalled.

**<expN1>** in syntax 1 is numeric value specifying the Inkey() code to place into the keyboard buffer. When **<expN1>** is 0 (zero), KeyTime(0) return the current trigger status: .T. if the trigger is active and .F. otherwise. In this case, **<expC2>** can be omitted.

**<expC2>** is a time value in the "hh:mm:ss" syntax specifying the system time at or after KeyTime() is triggered. The valid range is "00:00:00" to "23:59:59".

**<expL3>** is a logical value specifying that the **<expN1>** key value should be added into the keyboard buffer, same as ADDITIVE clause of the KEYBOARD command. The default .F. value specifies that the **<expN1>** value is placed unconditionally and overwrites all type- ahead values in the keyboard buffer.

## **Returns:**

**<retL>** is .T. on success, or .F. if the KeyTime() installation or de-installation fails.

## **Description:**

This function place specified Inkey() value into the keyboard buffer at an indicated time. It also allows to automatically place data into the keyboard buffer and (indirectly) execute user defined function at specified time.

In conjunction with SET KEY or ON KEY, you may perform any user action once the **<expN1>** key is removed from the buffer by Inkey(), InkeyTrap(), WAIT or READ. Note this difference to the TriggerUdf() function which is invoked directly, without the detour via keyboard buffer and hence without requirement of Inkey\*() or wait state.

The specified **<expN1>** value is placed into the keyboard buffer every time the given time value **<expC2>** occurs, or at the manual invocation via KeyTime(.T.). You will need to invoke KeyTime(.T.) from time to time in UDFs written in C, where the automatic trigger is inactive. The function run forever until the application ends, a new KeyTime() trigger is installed, or until de-installing it by invoking KeyTime(.F.) or KeyTime() without parameter.

Note that the use of KeyTime() may slow-down the execution speed. KeyTime() may coexist with other triggers like ShowTime(), KeySec(), and TriggerUdf().

Also note, that in the multi-user and multi-tasking environment, there is no guarantee for an exact time triggering, since other applications are executed parallel (in time slices) which may have higher priority (like system or kernel tasks) and may receive larger time slice. Therefore the exact definition of KeyTime() is "execute at or after the given time value".

As opposite to Clipper/CT3, you don't need to de-activate KeyTime() before the program ends, FS2 do it automatically.

**Example:**

This example demonstrates the combination of KeyTime(), ShowTime() and TriggerUdf(). It displays a time within the user should perform his entry (here in 30 seconds), the current system time, and beeps for the last 10 remaining seconds in 1-second interval.

```
local cVar := space(20), nTime, cTime
SET KEY K_CTRL_F20 TO myAbortRead // invoke this UDF via Ctrl-F20
nTime := seconds() + 30           // in 30 seconds from now on
cTime := SecToTime(nTime)         // convert to e.g. "13:15:36"
if !KeyTime(K_CTRL_F20, cTime)    // trap Ctrl-F20 after 30 sec.
    alert("could not install key trapping")
endif
TriggerUdf({|| myTrigger(nTime -10)}, 1) // trigger UDF every sec

@ 1,0 SAY "Enter data + press Return latest at " + cTime + " ("
xCol := col()                      // ShowTime() display coord
?? "00:00:00)"
@ row(), col() +1 GET cVar
ShowTime(row(), xCol)              // display current time
READ                               // process user entry or trap

KeyTime()                         // uninstall KeyTime()
ShowTime()                        // uninstall ShowTime()
TriggerUdf()                      // uninstall TriggerUdf()
SET KEY K_CTRL_F20 TO             // uninstall SET KEY

@ 3,1 say "read value = '" + cVar + "'"
wait "done..."
```

```
FUNCTION myAbortRead(p1,p2,p3)    // called via SET KEY from
KeyTime()
local oGet := GetActive()
if valtype(oGet) == "O" .and. !empty(oGet)
    oGet:VarPut( padr("aborted", len(oGet:Buffer) ) )
    CLEAR GETS
endif
return
```

```
FUNCTION myTrigger(nStartSec)     // called from TriggerUdf()
local nSec := Seconds()
if nSec >= nStartSec
    ?? chr(7)                     // beep in 1 sec interval
endif                             // for the last 10 seconds
return NIL
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools which supports only the first two arguments or no parameters at all to disable.

**Related:**

FS2:TriggerUdf(), FS2:KeyTime(), FS2:ShowTime(), Seconds(), Inkey(), InkeyTrap(), KEYBOARD, SET TYEAHEAD

# LASTDAYOM ()

---

**Syntax:**

**retN = LastDayOm ( [expDN1] )**

**Purpose:**

Determines the number of days in a month.

**Options:**

<expDN1> is either the date value of which the month is processed, or a numeric value (1..12) specifying the month within the current year. If not specified, the current month() of system date() is used.

**Returns:**

<retN> is the number of days in the month specified by <expDN1>, or 0 on error

**Description:**

LastDayOm() allows you to determine the numeric value of last day of the month that contains <expD1>.

Unlike EoM() which return date value, this LastDayOm() returns a numeric value (the number of days in the month).

**Example:**

```
? date()           // 03/22/2002
? LastDayOm()      // 31
? LastDayOm(2)     // 28
? LastDayOm(ctod("02/01/2000")) // 29
? LastDayOm(ctod("13/01/2000")) // 0
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:EoM(), Date(), DateValid(), Month(), Day(), Year(), Dtoc(), Dtos(), Ctod(), Stod()

# MDY ()

---

## Syntax:

**retC** = MDY ( [expD1] , [expL2] )

## Purpose:

Returns a date in "Month DD, [YY]YY" format.

## Options:

<expD1> is the date value that is processed. If not specified, the system date() is used.

<expL2> is a logical value: .T. specifies to use month names defined in FSsortab.\* and assigned by standard FlagShip function Cmonth(); .F. or NIL specifies to use month names defined in the source file fs2\_udfs.prg, same as Cmonth2()

Note: If the <expD1> is logical, a parameter shift is assumed and <expD1> is treated as <expL2>. Such a parameter shift is a bad programming technique and not suggested, but handled by FS2 for compatibility purposes to CT3.

## Returns:

<retC> is the resulting string in "Month DD, [YY]YY" format, or "" if the function fails.

## Description:

Use MDY() to return the date as a string using the name of month, day, and year. The SET CENTURY ON/OFF switch determines whether the year is displayed in two or four digits.

For a full internationalization, this function is available in source code in the fs2\_udfs.prg. The default language is English, but you can modify the months (in the array at the beginning of the source code) to fit to your native language.

## Example:

```
SET CENTURY OFF
? MDY() // "March 7, 03"
? MDY( ctod("12/31/02") ) // "December 31, 02"
? MDY( ctod("13/31/02") ) // ""

lCentury := set(_SET_CENTURY, .T.) // same as SET CENTURY ON

FS_SET ("load", 1, "FSsortab.ger")
FS_SET ("setl", 1)
? MDY() // "March 7, 2003"
? MDY(.T.) // "Maerz 7, 2003"
? MDY( ctod("12/01/89") ) // "December 1, 1989"
? MDY( ctod("12/01/89"), .T. ) // "Dezember 1, 1989"
? Cdw2() + " " + MDY() // "Friday March 7, 2003"
? Cdw() + " " + MDY(.T.) // "Freitag Maerz 7, 2003"

set(_SET_CENTURY, lCentury) // reset current CENTURY
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools does not support modifiable internationalization.

**Related:**

FS2:DMY(), FS2:Cdow2(), FS2:Cmonth2(), FS2:NtoCmonth(), FS2:NtoCdow(), Date(), WebDate(), Month(), Day(), Year(), Dtoc(), Dtos(), Ctod(), Cdow(), Cmonth()

# NTOCDOW ()

---

**Syntax:**

**retC** = NtoCdw ( [expN1] )

**Purpose:**

Converts the number of a weekday into a weekday name.

**Arguments:**

<expN1> is a weekday number from 1 to 7. If not specified, the current weekday number is determined from the system date().

**Returns:**

<retC> is a character string, containing the name of the day of the week (e.g. "Monday" or "Montag"), or "" on error.

**Description:**

This function converts a weekday number into the corresponding name. Sunday is 1, Monday is 2, Saturday is 7.

Unlike the similar Cdw() and Cdw2() which accepts date value, this NtoCdw() determines the day from the passed weekday number.

For a full internationalization, this function is available in source code in the fs2\_udfs.prg. The default language is English, but you can modify the day names (in the array at the beginning of the source code) to fit to your native language.

**Example:**

```
? NtoCdw()           // "Monday"
? NtoCdw()           // "anything-else" (if changed)

? NtoCdw(1)          // "Sunday"
? NtoCdw(7)          // "Saturday"
? NtoCdw(8)          // ""

? Cdw2()              // "Monday"
? Cdw2(date() + 2)    // "wednesday"
? Cdw2()              // "anything-else" (if changed)
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools does not support modifiable internationalization.

**Related:**

FS2:Cdw2(), FS2:Cmonth2(), FS2:CtoMonth(), Cdw(), Dow(), Date(), Month(), Day(), Year(), Dtoc(), Dtos()



# NTOCMONTH ()

---

**Syntax:**

**retC** = NtoCmonth ( [expN1] )

**Purpose:**

Converts the number of a month into a month name.

**Arguments:**

<expN1> is a month number from 1 to 12. If not specified, the current month number is determined from the system date().

**Returns:**

<retC> is a character string, containing the name of the month (e.g. "March" or "M,,rz"), or "" on error.

**Description:**

This function determines the month name that corresponds to the given month number. January is 1, February is 2, December is 12.

Unlike the similar Cmonth2() which accepts date value, this NtoCmonth() determines the month name from the passed month number.

For a full internationalization, this function is available in source code in the fs2\_udfs.prg. The default language is English, but you can modify the month names (in the array at the beginning of the source code) to fit to your native language.

**Example:**

```
? NtoCmonth()           // "March"
? NtoCmonth()           // "anything-else" (if changed)

? NtoCmonth(1)          // "January"
? NtoCmonth(12)         // "December"
? NtoCmonth(13)         // ""

? Cmonth2()             // "March"
? Cmonth2(date() + 60)  // "May"
? Cmonth2(ctod("08/20/02")) // "August"
? Cmonth2()             // "anything-else" (if changed)
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools does not support modifiable internationalization.

**Related:**

FS2:NtoCdw(), FS2:Cmonth2(), FS2:CtoMonth(), Cmonth(), Dow(), Date(), Month(), Day(), Year(), Dtoc(), Dtos()

# QUARTER ()

---

**Syntax:**

**retN = Quarter ( [expD1] )**

**Purpose:**

Determines the quarter in which a specific date lies.

**Options:**

<expD1> is the date value that is processed. If not specified, the system date() is used.

**Returns:**

<retN> is a number that identifies the quarter (1...4) in which the <expD1> date lies, or 0 if the function fails.

**Description:**

Quarter() determines the quarter in which a specific date lies.

**Example:**

```
? date() // 03/22/2002
? Quarter() // 1
? Quarter( AddMonth(3) ) // 2 (next quarter)
? Quarter( ctod("09/15/02") ) // 3
? Quarter( ctod("13/15/02") ) // 0 (error)
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

Date(), Month(), Day(), Year(), Dtoc(), Dtos(), Ctod(), Stod()

# SECTOTIME ()

---

**Syntax:**

```
retC = SecToTime ( [expN1], [expL2] )
```

**Purpose:**

Converts seconds into a time string.

**Options:**

<expN1> is the number of seconds to convert into a character string in time format. If not specified, the current seconds() is used and is then comparable to the Time() output. The <expN1> value can be positive, zero or negative.

<expL2> is an output mode (default is .F.). If specified .T., the resulting string contains hundredths of seconds. The fraction of seconds is calculated only when the number of days is less than 230.

Note: If the <expN1> is logical, a parameter shift is assumed and <expN1> is treated as <expL2>. Such a parameter shift is a bad programming technique and not suggested, but handled by FS2 for compatibility purposes to CT3.

**Returns:**

<retC> is the resulting string in format HH:MM:SS" or "HH:MM:SS.ff" or "D[DDDD]:HH:MM:SS[.ff]" if the result exceed 24 hours. The seconds (and decimals) values are not rounded but truncated. On error, "" is returned.

**Description:**

This function can be applied in two areas - to convert current time to seconds, and to convert a time period into the "HH:MM:SS" or "HH:MM:SS.hh" format, comparable to the Time() output.

When the <expN1> is equal to or exceeds 24 hours (86400 seconds), the output include days in the form "ddd::hh:mm:ss[.ff]".

**Example:**

```
? nStart := Seconds()           // 10170.0
... long process...
? nEnd := Seconds()             // 13656.8
? "operation took", SecToTime(nEnd - nStart), "h:m:s"
                                // "operation took 00:58:06 h:m:s"

? SecToTime(45873.123, .T.)      // "12:44:33.12"          12 hours
? SecToTime(86400 *20 +1)        // "20::00:00:01"          20 days
? SecToTime(-86400 *200, .T.)    // "-200::00:00:00.00"     -200 days
? SecToTime(2049840000)          // "23725::00:00:00"     ca 65 years

? SecToTime(Seconds())           // "16:54:05"
? SecToTime()                    // "16:54:05"
? SecToTime(, .T.)               // "16:54:05.32"
? SecToTime(.T.)                 // "16:54:05.32"

? Time()                         // "16:54:05"
```

```
? Time(1) // "16:54:05.3207"  
? Time(2) // "31-10-02 16:54:05.3209"  
? Time(3) // "02-10-31 16:54:05.3212"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools requires the first optional argument and support values < 86400 i.e. up to 24 hours only. For CT3 compatible output of values larger than 24 hours, use SecToTime(value % 86400) which then discards the days formatting.

**Related:**

FS2:TimeToSec(), FS2:TimeValid(), Seconds(), Time()

# SETDATE ()

---

**Syntax:**

`retL = SetDate ( expD1, [expL2] )`

**Purpose:**

Sets the system date.

**Arguments:**

<expD1> is the date value that is to set as new system date.

**Options:**

<expL2> is ignored in FS2. It is used in CT3/DOS to specify whether the CMOS RAM should be affected too.

**Returns:**

<retL> is .T. if the new date could be set, or is .F. if the function fails.

**Description:**

With SetDate() you can change the system date.

Note: In multiuser and/or multitasking systems, you need to have sufficient permission to set system date. In Unix/Linux, the current user must have root or su rights. On MS-Windows, the user must have administrator permission. If not so, the function fails, same as attempt to change date/time at console prompt would.

**WARNING:** the system date and system time are crucial settings, very often used by the operating system self, by daemons (services) and by other applications concurrently running. Changing the system date (or time) may cause system desynchronization resulting in malfunction or in hang of the current or other applications, X11 or window manager, "blue screen" in Windows and so on. So be very careful and think twice about before using this function! Common is to set the date/ time at system basis by the responsible administrator and before other applications are started.

**Example:**

```
if ! SetDate( date() +1 )      // system date = tomorrow
    alert("sorry, insufficient permission to set system date")
endif
? date()
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:SetTime(), Date(), Ctod()

# SETTIME ()

---

**Syntax:**

`retL = SetTime ( expC1, [expL2] )`

**Purpose:**

Sets the system time.

**Arguments:**

<expC1> is the time value that is to set as new system date. The valid format is "hh:mm:ss" or "hh:mm".

**Options:**

<expL2> is ignored in FS2. It is used in CT3/DOS to specify whether the CMOS RAM should be affected too.

**Returns:**

<retL> is .T. if the new time could be set, or is .F. if the function fails.

**Description:**

With SetTime() you can change the system time.

Note: In multiuser and/or multitasking systems, you need to have sufficient permission to set system time. In Unix/Linux, the current user must have root or su rights. On MS-Windows, the user must have administrator permission. If not so, the function fails, same as attempt to change date/time at console prompt would.

**WARNING:** the system date and system time are crucial settings, very often used by the operating system self, by daemons (services) and by other applications concurrently running. Changing the system date (or time) may cause system desynchronization resulting in malfunction or in hang of the current or other applications, X11 or window manager, "blue screen" in Windows and so on. So be very careful and think twice about before using this function! Common is to set the date/ time at system basis by the responsible administrator and before other applications are started.

**Example:**

```
if ! SetTime("14:05:26")      // new system time = 2.05pm
    alert("sorry, insufficient permission to set system time")
endif
? date(), time()
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:SetDate(), Time()

# SHOWTIME ()

---

## Syntax 1:

```
retL = ShowTime ( [expN1], [expN2], [expL3],  
                  [expCN4], [expL5], [expL6] )
```

## Syntax 2:

```
retL = ShowTime ( [expL1] )
```

## Syntax 3:

```
retL = ShowTime ( expN1 )
```

## Purpose:

Continuously displays the time at desired screen position.

## Options:

<expL1> in syntax 2 is an optional logical parameter. If specified .T., the time is displayed enforced, i.e. independent of the current time value. If specified .F. or no parameters are given, ShowTime() display is uninstalled.

<expN1> in syntax 1 is the row where the time is displayed, either in row/col or in pixel in dependence of the current SET PIXEL status. If not given, the default is 0. When <expN1> is 0 and no additional parameters are given according to syntax 3, ShowTime(0) return the current status: .T. if ShowTime() is active and .F. otherwise.

<expN2> is the column where the time is displayed, either in row/col or in pixel in dependence of the current SET PIXEL status. If not given, the default is 0.

<expL3> is a logical value specifying whether the display of seconds should be suppressed. The default is .F. designating to display seconds.

<expCN4> is the color attribute used to display the time. Either standard SET COLOR string ("foreground/background") or a numeric value corresponding to the FS2:NtoColor() parameter is accepted. If not given, the current color, determined via SetColor() at the time of ShowTime() initialization, is used.

<expL5> is a logical value specifying whether the display should be in 12 hour format. The default is .F. designating to display the time in 24hr format.

<expL6> specifies whether the "a" and/or "p" should be appended to the display. This is often used in conjunction with set <expL5>. The default is .F. designating to display it without a/p suffix.

## Returns:

<retL> is .T. on success and .F. on error

## Description:

In FlagShip, the time display is already integrated in the status bar automatically. This status bar is active in GUI and can be activated in Terminal i/o as well.

However, you may use this ShowTime() function to constantly display the time in any screen position desired. Note that the use of ShowTime() may slow-down the execution speed.

ShowTime() is supported in GUI and in Terminal i/o mode. In the basic i/o mode, the ShowTime() call is ignored. ShowTime() is triggered approximately every second. You may force the display manually by invoking ShowTime(.T.). ShowTime() may run parallel to KeySec(), KeyTime(), and TriggerUdf().

As opposite to Clipper/CT3, you don't need to de-activate ShowTime() before the program ends, FS2 do it automatically.

**Example:**

```
ShowTime(0,20)           // ShowTime() is active on the top
wait
ShowTime()               // de-activates ShowTime()
@ 0,20 clear to 0,40

ShowTime(maxrow(),60,,"w+/b",.T.,.T.) // ShowTime() at the bottom
@..GET..
READ

nn := 0
? "processing loop"
for ii := 1 to 500000
    // ...your statements, e.g.
    nn++
    if (nn % 10000) == 0
        ?? "."                // display loop progress
    endif
next
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools which support only syntax 1 or no parameters at all to disable.

**Related:**

FS2:TriggerUdf(), FS2:KeySec(), FS2:KeyTime(), StatusBar class



# TIMETOSEC ()

---

**Syntax:**

**retN = TimeToSec ( [expC1], [expL2] )**

**Purpose:**

Convert the character time value to seconds since midnight.

**Options:**

<expC1> is the time value that is converted. Accepted time format is "hh" or "hh:mm" or "hh:mm:ss" or "hh:mm:ss.ff" or "hh:mm:ss:ff" where hh = hours (0..23), mm = minutes (0..59), ss = seconds (0..59) and ff = fractions of seconds (0..9999). You may optionally preface the time value by days in the syntax "[dd:]hh[:mm:ss:ff]" where dd = days (0..65535). Leading and trailing spaces are ignored, same as inserted characters between the numeric value and colon separator. The string may not start nor end with separator. Missing values are assumed to be 0.

If <expC1> is not specified, the current time is used and the output is then equivalent to the standard function Seconds().

<expL2> if specified .T., a time value "24:00:00" is accepted as well except with days. Otherwise the upper time limit is "23:59:59.9999"

**Returns:**

<retN> is the result of the time conversion to seconds, or -1 on error.

**Description:**

TimeToSec() converts the character based time value, e.g. from the Time() function or from user input, into numeric value representing seconds since midnight (except when the day is given, where the return value represents total seconds of the time character value). The resulting numeric value can be compared to or calculated with other second values, coming e.g. from the standard FlagShip function Seconds().

**Example:**

```
? time(), time(1) // "13:54:48" "13:54:48.2615"
? TimeToSec( time() ), TimeToSec() // 50088.00 50088.26

? nBeg := Seconds() // 50088.26
? cBeg := time() // "13:54:48"
inkey(2)
? Seconds() - TimeToSec(cBeg) // 2.26
? int(Seconds()) - TimeToSec(cBeg) // 2.00
? Seconds() - nBeg // 2.00

? TimeToSec("") // 0.00
? TimeToSec(space(20)) // 0.00
? TimeToSec(" : :1") // 1.00
? TimeToSec(" : : :5") // 0.50
? TimeToSec("0:0:1.5") // 1.50

? TimeToSec("23") // 82800.00
```

```

? TimeToSec("23:") // -1
? TimeToSec("23: ") // 82800.00
? TimeToSec("23:60") // -1
? TimeToSec("23:59") // 86399.00
? TimeToSec("24") // -1
? TimeToSec("24", .T.) // 86400.00
? TimeToSec("24:0:1", .T.) // -1
? TimeToSec(" 23: 59 : 59 .99") // 86399.99
? TimeToSec(" 23h:59m:59s:99") // 86399.99

? TimeToSec(":5") // -1
? TimeToSec(" :5") // 300.00
? TimeToSec(" ::5") // -1
? TimeToSec(" :: :5") // 300.00
? TimeToSec(" :: : : ") // 0.00
? TimeToSec(" :: : :5") // 5.00
? TimeToSec("1::0:0:5") // 86405.00
? TimeToSec("-1::0") // -86400.00

SET DECIMALS TO 4
? TimeToSec("12") // 43200.0000
? TimeToSec("12:44") // 45840.0000
? TimeToSec("12:44:33") // 45873.0000
? TimeToSec("12:44:33:22") // 45873.2200
? TimeToSec("12:44:33.2345") // 45873.2345
? TimeToSec("100::12:44:33.2345") // 8685873.2345
? SecToTime(8685873.2345, .T.) // "100::12:44:33.23"

```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CT3 does not support the second parameter, requires all values to be two digits and does not accept spaces nor additional characters.

**Related:**

FS2:SecToTime(), FS2:TimeValid(), Time(), Seconds()

# TIMEVALID ( )

---

## Syntax:

```
retL = TimeValid ( [expC1], [expL2] )
```

## Purpose:

Determines whether the character time value is valid and can be converted to seconds via TimeToSec().

## Options:

<expC1> is the time value that is converted. Accepted time format is "hh" or "hh:mm" or "hh:mm:ss" or "hh:mm:ss.dd" or "hh:mm:ss:ff" where hh = hours (0..23), mm = minutes (0..59), ss = seconds (0..59) and ff = fractions of seconds (0..9999). You may optionally preface the time value by days in the syntax "[ddd:hh:mm:ss:ff]" where dd = days (0..65535). Leading and trailing spaces are ignored, same as inserted characters between the numeric value and colon separator. The string may not start nor end with separator. Missing values are assumed to be 0.

If <expC1> is not specified, the current time is used and hence the time value is valid.

<expL2> if specified .T., a time value "24:00:00" is accepted as well except with days. Otherwise the upper time limit is "23:59:59.9999"

## Returns:

<retL> is .T. if the given character time value is valid and can be converted to seconds via TimeToSec(), or is .F. on failure.

## Description:

TimeValid() tests whether the character based time value, e.g. from the Time() function, can be converted via TimeToSec() into numeric value representing seconds since midnight. TimeValid() can also be used for validation of time input values.

## Example:

```
? TimeValid()                // .T.    (current time)
? TimeValid("")              // .T.    (zero time)
? TimeValid(" ")            // .T.    (zero time)
? TimeValid("::")           // .F.
? TimeValid(" : ")          // .T.    (zero time)
? TimeValid(" : : ")        // .T.    (zero time)
? TimeValid(" : :0")        // .T.    (zero time)
? TimeValid(" : : : ")      // .T.    (zero time)
? TimeValid("0::0:0:0.0")    // .T.    (zero time)

? TimeValid("23:60")         // .F.
? TimeValid("24")            // .F.
? TimeValid("24", .T.)       // .T.
? TimeValid("24:0:1", .T.)   // .F.
```

```

? TimeValid(" 23: 59 : 59 .99")      // .T.
? TimeValid(" 23h:59m:59s:98")      // .T.
? TimeValid("100days::23h:59m:59s")  // .T.

? TimeValid("12")                    // .T.
? TimeValid("12:44")                 // .T.
? TimeValid("12:44:33")              // .T.
? TimeValid("12:44:33:22")           // .T.
? TimeValid("12:44:33.2345")         // .T.
? TimeValid("3::12:44:33.2345")      // .T.

cBeg := cEnd := space(5)
@ 1,10 SAY "work begin" GET cBeg PICT "99:99" ;
      VALID TimeValid(cBeg) .and. TimeToSec(cBeg) > 0
@ 2,10 SAY "work done " GET cEnd PICT "99:99" ;
      VALID TimeValid(cEnd) .and. ;
      (TimeToSec(cEnd) - TimeToSec(cBeg)) > 0
READ
? "spent", SecToTime(TimeToSec(cEnd) - TimeToSec(cBeg)), "h:m:s"

```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CT3 does not support the second parameter, requires all values to be two digits and does not accept spaces nor additional characters.

**Related:**

FS2:TimeToSec(), FS2:SecToTime(), Time(), Seconds(), DateValid()

# TRIGGERUDF ()

---

## **Syntax1 :**

**retL** = TriggerUdf ( **expCB1**, [**expN2**] )

## **Syntax 2:**

**retL** = TriggerUdf ( [**expL1**] )

## **Syntax 3:**

**retL** = TriggerUdf ( **expN1** )

## **Purpose:**

Triggers a user defined function (or code block) each second or each specified time value.

## **Arguments:**

<**expL1**> in syntax 2 is an optional logical parameter. If specified .T., the TriggerUdf() function is triggered enforced, i.e. independent of the current time value. If specified .F. or no parameters are given, TriggerUdf() is uninstalled.

<**expN1**> in syntax 3 accepts only numeric 0 (zero). It then return the current trigger status: .T. if the trigger is active and .F. otherwise.

<**expCB1**> in syntax 1 is either character value specifying the name of UDF, or a codeblock. Either the UDF or the code block is then executed every <expN2> seconds.

<**expN2**> is numeric value specifying the time interval in seconds after which TriggerUdf() is triggered. The valid range is 1 to 86400 (i.e. 1sec to 24hr). If not specified, 1 sec. is assumed.

## **Returns:**

<**retL**> is .T. on success, or .F. if the TriggerUdf() installation or de-installation fails.

## **Description:**

TriggerUdf() is similar to KeySec(), but the specified UDF or codeblock is triggered directly, without detour via SET KEY and is hence active also without wait state.

The codeblock receives three arguments corresponding to ProcName(), ProcLine() and Seconds(), the UDF specified as string is called w/o parameters.

The triggering action run forever as long as the application executes or until it action is disabled by TriggerUdf([.F.]) or until a new TriggerUdf() action is initialized. To avoid endless recursion (e.g. when the UDF needs longer than the trigger time value), the next UDF or codeblock call occurs only when the previous call was finished. You will need to invoke TriggerUdf(.T.) from time to time in UDFs written in C, where the automatic trigger is inactive.

Note that the use of TriggerUdf() may slow-down the execution speed. TriggerUdf() may coexist with other triggers like KeySec(), KeyTime(), and ShowTime().

Also note, that in the multi-user and multi-tasking environment, there is no guarantee for an exact time triggering, since other applications are executed parallel (in time slices) which may have higher priority (like system or kernel tasks) and may receive larger time slice. Therefore the exact definition of TriggerUdf() is: "execute at or after the given time value expires".

**Example:**

```
TriggerUdf( {|name,line,sec| qqout(7) }, 5) // beep every 5
seconds
```

**Example:**

```
if !TriggerUdf("myudf")           // call myUdf() every second
    Alert("Could not install trigger")
endif
? "Trigger is " + if(TriggerUdf(0),"","NOT ") + "active"
// ...
TriggerUdf(.F.)                   // uninstall it

FUNCTION myUdf()
    // do anything repeatedly ...
return
```

**Example:**

```
LOCAL nStart := Seconds()
TriggerUdf( {|| MyUdf(nStart) }, 2 ) // call myUdf() every 2
seconds
// ...

FUNCTION myUdf(nStart)
    // do anything repeatedly ...
    if Seconds() - nStart > 3600
        TriggerUdf(.F.)           // uninstall itself after 1 hour
    endif
return
```

**Example:**

```
See additional example with READ in the FS2:KeyTime() description
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox. Not available in NT2/CA3 tools.

**Related:**

FS2:KeySec(), FS2:KeyTime(), FS2:ShowTime(), Seconds()

# WAITPERIOD ()

---

**Syntax:**

```
retL = WaitPeriod ( [expN1], [expL2] )
```

**Purpose:**

Checks if specified time period is expired.

**Options:**

<expN1> is the time period either in 1/100 seconds (when <expL2> is not given or if .F.) or in seconds (if <expL2> is .T.) which the subsequent WaitPeriod() call should test.

<expL2> is a modifier for the period value. If .F. (the default), the <expN1> value represents 1/100 seconds (e.g. 150 is 1.5 sec). If given .T., the <expN1> value represents second value including its decimal fraction.

**Returns:**

<retL> is .T. when the current system time is within the specified <expN1> span, or is .F. if the wait period has elapsed or is not set.

**Description:**

This function allows you to limit loops which otherwise would run endless. You initialize the maximum time span and then call this function without parameters; it returns .F. when the period has elapsed. It is similar to check Seconds() at the beginning and during the execution but avoids its calculation.

Once the wait period expires, you need to set it anew for further use.

Note: to avoid heavy system load in large endless loop, preferably use sleep(n) or sleepms(n) or inkey(n) to allow other processes to execute too.

**Example:**

```
nNum := 0
? "running a loop for max. 10.5 sec or 100 mio steps"
? time(1)                // 15:13:00.3650

waitPeriod(1050)          // initialization, 10.5 seconds
* waitPeriod(10.5, .T.)   // alternative syntax

while waitPeriod() .and. nNum < 100000000
    nNum++
    // sleepms(500)        // sleep for 0.5 sec
enddo

? time(1), nNum           // 15:13:10.8650   2959263
```

**Classification:** add-on library, programming

**Compatibility:** Available in FS2-Toolbox, compatible to NT2/CA3 tools. CT3 supports only the first argument, and hence does not accept input in seconds but in 1/100 seconds only.

**Related:** Seconds(), Time()

# WEEK ()

---

**Syntax:**

**retN = Week ( [expD1] , [expL2] )**

**Purpose:**

Returns the calendar week for a date. The week always starts on a Monday.

**Options:**

<expD1> is the date value that is processed. If not specified, the current system date() is used.

<expL2> is the compatibility switch for ISO 8601. If .T., WEEK() considers ISO 8601 (see below). If .F., the year always starts with week 1 and ends with week 52 or 53. If <expL2> is NIL or not given, defaults set by WeekSetIso(expL) are used, where the default is .T. = ISO 8601.

**Returns:**

<retN> is the sequential week number (1..53) that designates in which week of the year <expD1> lies, or 0 on error.

**Description:**

The week of the year in which a day lies is information required by a number of technical financial calculations or wage accounting. This function determines this week and acknowledged all calendar rules:

When <expL2> is .F. or WeekSetIso(.F.) was set, WEEK() always return absolute weeks, i.e. 01-Jan-Year is week 1 and 31-Dec-Year is week 52 or 53.

When <expL2> is .T. or WeekSetIso(.T.) was set (the default), WEEK() considers the ISO-8601 week-of-year handling. This means, 01-Jan-Year may return week 52, 53 or 1 in dependence on the calendar day, see below. Similarly, 31-Dec-Year may return week 52, 53 or 1.

ISO-8601 says: A week is a seven-day period within a calendar year, starting on a Monday and identified by its ordinal number within the year; the first calendar week of the year is the one that includes the first Thursday of that year. In the Gregorian calendar, this is equivalent to the week which includes 4 January. First week of year should show Week with a minimum of 4 days. Week 1 will then always include at least a Thursday, Friday, Saturday and Sunday.

Note that, by ISO standard, if January 1 is on a Friday, Saturday or Sunday, this short week is counted as part of the last week of the previous year and is displayed by Week() as week 52 or 53. Similarly if January 1 is on a Tuesday, Wednesday or Thursday, the last few days of the previous year are displayed as week 1.



**Example:**

```
set date american
? week() // 11
? week( ctod("05/03/02") ) // 18
? week( ctod("02/29/02") ) // 0
? week( ctod("02/29/04") ) // 9
? week( ctod("12/31/04") ) // 53

? week( ctod("01/02/05"), .T.) // 53
? week( ctod("01/02/05"), .F.) // 1
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:WeekSetIso(), FS2:WoM(), FS2:Quarter(), FS2:IsLeap(), FS2:CtoDow,  
FS2:NtoCdw(), FS2:NtoCmonth(), FS2:Cdw2(), Cdw(), DateValid(), Date(),  
Day(), Month(), Year(), Ctod(), Stod()

# WEEKSETISO ()

---

**Syntax:**

**retL = WeekSetIso ( [expL1] )**

**Purpose:**

Returns or sets the default calendar week() behavior.

**Options:**

<expL1> is the compatibility switch for ISO 8601. If .T., WEEK() considers ISO 8601, see the WEEK() description for details. If .F., the year always starts with week 1 and ends with week 52 or 53. If <expL1> is not given, WEEKSETISO() only returns the current setting.

**Returns:**

<retL> is the current setting (at the time of entering this function).

**Description:**

The WEEK() function handles two different modes: absolute weeks starting with 1 at 01/01/year, or calendar weeks conformable with ISO 8601, where the first week may be 52 or 53. The behavior is controlled by WeekSetIso(), or by the second parameter of Week(). The default setting is .T. which considers ISO 8601 calendar handling.

**Example:**

```
? WeekSetIso()           // .T.
? week( ctod("01/01/05") ) // 53
? WeekSetIso(.F.)        // .T.
? WeekSetIso()           // .F.
? week( ctod("01/01/05") ) // 1
? week( ctod("01/01/05"), .T. ) // 53
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, not available in NT2/CA3 tools.

**Related:**

FS2:Week()

# WOM ()

---

**Syntax:**

**retN = WOM ( [expD1] )**

**Purpose:**

Returns the week of a month.

**Options:**

<expD1> is the date value that is processed. If not specified, the current system date() is used.

**Returns:**

<retN> is the sequential week number (1..6) that reflects the week within a month in which <expD1> lies, or 0 on error. A week starts on a Monday.

**Description:**

The week of the month is information required by a number of technical financial calculations or wage accounting.

**Example:**

```
? WOM() // 3
? WOM( ctod("05/03/02") ) // 1
? WOM( ctod("02/29/02") ) // 0
? WOM( ctod("02/29/04") ) // 5
? WOM( ctod("12/31/04") ) // 5
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:Week(), FS2:Quarter(), FS2:IsLeap(), FS2:CtoDow, FS2:NtoCdw(), FS2:NtoCmonth(), FS2:Cdw2(), Cdw(), DateValid(), Date(), Day(), Month(), Year(), Ctod(), Stod()

# Windowing Functions

---

## Introduction

The Windowing functions allows you to create any number of additional sub-windows for both Terminal and GUI i/o mode. These sub-windows of specified size overlay the main screen and can be moved on the main screen both programmatically and by user using cursor keys. You may use any FlagShip i/o functions or class to display and/or input data on this sub-window. In GUI mode, you may specify at the time of invoking Wopen() that the sub-window is created as MDI screen instead of additional semi-modal SDI window.

As opposite to Clipper, the standard functions like MaxCol(), MaxRow(), Col(), Row() etc. automatically consider the FS2 Toolbox sub-windows and don't need to load any additional drivers or objects.

### Index of FS2 Windowing Functions

WaClose()	Closes all open sub-windows and select main screen
Wboard()	Allocates allowable screen area for sub-windows
Wbox()	Draw frame box around window
Wcaption()	Write textual caption in the window frame
Wcenter()	Center the specified or current window
Wclose()	Close the current or specified sub-window
Wcol()	Column number of left sub-window frame
WfCol()	Position of the leftmost column of formatted area
WfLastCol()	Position of the rightmost column of formatted area
WfLastRow()	Position of the bottom row of formatted area
Wformat()	Specifies the usable area within a sub-window
WfRow()	Position of the top row of the formatted area
WlastCol()	Column number of right sub-window frame
WlastRow()	Row number of bottom sub-window frame
WmaxCol()	Max available column
WmaxRow()	Max available row
Wmiddle()	Center the current sub-window
Wmode()	Turns the screen border overstep mode
Wmove()	Moves the current or specified sub-window
WmoveDown()	Move sub-window down number of rows set by Wstep()
WmoveLeft()	Move sub-window left number of columns set by Wstep()
WmoveRight()	Move sub-window right number of columns set by Wstep()
WmoveUp()	Move sub-window upwards number of rows set by Wstep()
WmoveUser()	Enable/disable window movement by cursor keys
Wnum()	Determines the highest window ID#
Wopen()	Open a new sub-window
Wrow()	Row number of upper sub-window frame

Wselect()	Activates one of the open sub-windows or the main screen
WsetMove()	Turns the interactive user movement mode on or off
WsetShadow()	Sets the window shadow colors
Wstep()	Set the step width of interactive window movement

## Programming with FS2 Window Functions

Each window is assigned a unique positive number when it is created and opened via Wopen(). This number is known as the window ID# (or as window handle). This ID# is used for handling and manipulation of sub-windows, it is therefore important to save the ID# to a variable when you open it, so that you can select the required window later.

The last opened window will automatically be selected and become input focus. The ID# 0 has special meaning: it is the ID of your main screen, and you can switch between any of the sub-windows and the main screen.

```
nwin1 := wopen(...)
nwin2 := wopen(...)           // This is the active window
...
wselect(nwin1)                // Activate first sub-window
...
wselect(0)                    // Activate main screen
...
wselect(nwin2)                // Activate second sub-window
```

A sub-window is a virtual screen in terminal i/o, or is a separate widget (control) in GUI. From programmer's view, it differs from the main screen only in size. In GUI mode, you can specify the appearance of the sub- window: either an additional MDI window, or an additional modal or semi- modal window; see further details below in Wopen() reference.

The underlying screen area is saved automatically when a new window is created. This applies equally to any area of the main screen that becomes overlapped by the movement of a sub-window.

**Active Windows:** When you create a sub-window via Wopen(), this window is automatically selected and becomes input focus. You may select any other sub-window or the main screen via Wselect(). You may also retrieve or manipulate unselected windows by specifying the optional ID# parameter of the corresponding W\*() function. When you close the currently selected window via Wclose(), the sub-window with the highest ID# is selected. You can retrieve the active window ID# by calling Wselect() with no parameters. The active window is marked in GUI by [\*] sign in the header. In GUI mode, you also may select and view an inactive window (read-only) by Menu->Windows->Sub-Windows (user modifiable), see also initomenu.prg for details.

**Input/Output:** The coordinates used for screen output are relative only to the selected sub-window, and not the entire screen. You may use both the common row/col coordinates or pixels in GUI mode. Even the Col(), Row(), MaxCol() and MaxRow() standard functions apply for the currently selected sub-window or the main screen respectively.

```

nwin1 := wopen(10,10,20,40) // Activate and select sub-window

@ 2,2 SAY "FS2 Toolbox"      // and display text there
?? " is greeting you"

nCol   := Col()              // save current sub-window status
nRow   := Row()
nMaxCol := MaxCol()
nMaxRow := MaxRow()

wselect(0)                   // select main screen

? "The main screen is", ltrim(MaxRow()), "x", ltrim(MaxCol())
? "The first sub-window with ID#" + ltrim(nwin1),"is", ;
  ltrim(nMaxRow), "x", ltrim(nMaxCol)
? "and the cursor is there at",ltrim(nRow), "x", ltrim(nCol)
wait

wclose(nwin1)                // close sub-window, stay in ID#0

```

The sub-window behaves same as the main screen. All the standard settings, including SET PIXEL ON|OFF apply also for the selected sub-window. But local row/column status and SET COLOR settings apply for the currently selected sub-window only.

# WACLOSE ()

---

**Syntax:**

`retN = WaClose ()`

**Purpose:**

Closes all open sub-windows and select main screen.

**Returns:**

<retN> is 0 when successful or -1 if an error occurs

**Description:**

This function closes all open sub-windows previously created by Wopen(). If no sub-windows are open, then WaClose() returns a value of -1, otherwise 0 is returned.

**Example:**

```
nwin1 := wopen( ... )           // Many sub-windows will be opened
nwin2 := wopen( ... )
nwin3 := wopen( ... )

wclose(nwin2)                   // closes second sub-window
waclose()                       // closes sub-windows nwin1 and nwin3
```

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

Wopen(), Wclose()

# WBOARD ()

---

**Syntax:**

```
retN = Wboard ( [expN1], [expN2], [expN3],  
                [expN4] )
```

**Purpose:**

Allocates allowable screen area for sub-windows. Apply for Terminal i/o mode only, ignored otherwise.

**Arguments:**

<expN1> to <expN4> are optional numeric coordinates specifying the upper row, left column, bottom row and right column, in that order, within the sub-windows are available.

If a parameter is not specified or out of range 0..MaxRow()/MaxCol(), the default screen size value for this parameter is used.

**Returns:**

<retN> is 0 if all four parameters are given and correctly used, or is set to -1 otherwise.

**Description:**

This function defines the screen area where sub-windows are permitted. All sub-windows are subsequently only visible in this defined area. You can use this function to protect an area of the screen from being overwritten with sub-windows, even when you allow the user to move the window interactively. The boundaries designated by Wboard() become the screen boundaries for window functions.

Wboard() is only effective when there are no open windows and is considered in Terminal i/o mode only.

**Example:**

Leave only the upper half of the screen for window functions

```
waClose()  
wboard(0,0, MaxRow() / 2, MaxCol())
```

**Classification:**

add-on library, programming, windowing

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. Apply for Terminal i/o mode, ignored in GUI mode.

**Related:**

Wopen(), WaClose(), MaxRow(), MaxCol()



# WBOX ()

---

## Syntax:

**retN = Wbox ( [expCN1], [expC2], [expN3] )**

## Purpose:

Draw frame box around window, similarly to @..BOX command. Apply for Terminal i/o mode only.

## Arguments:

<expC1> is a character string with 8 or 9 characters specifying the border frame, same as in @..BOX command. You can specify an individual character with which to frame the entire window. You can specify up to nine different characters for the corners, and the horizontal and vertical lines. You can also specify a character with which to fill the window. The frame characters start to fill the frame in the upper left corner and continue clockwise around the frame. When specified, the ninth character is used to fill the window. When the clear character was set by SetClearBox(), it is used instead.

<expN1> is supported for CA3 toolbox compatibility only: a numeric value of the border type

If <expCN1> is not specified, the default value is taken from global variable \_aGlobSetting[GSET\_T\_C\_AT\_TO\_DOUBLE] defined in initio.prg

<expC2> is an optional string with color specification of the frame. Only the first color pair is used: the frame is drawn by its foreground/background, the box is filled by the background color. If not specified, the current SET COLOR is used.

<expN3> is the window ID# for which this function apply. If not specified, the currently selected sub-window is used.

## Returns:

<retN> is a numeric value representing the currently selected sub-window, or -1 on error.

## Description:

This function works like the @...BOX command, except that Wbox() always encloses the currently selected or the specified sub-window. You can specify an individual character with which to frame the entire window.

The available area for the window is reduced by two rows and two columns (i.e. Wformat(1,1,1,1) is automatically set). This prevents the border from being unintentionally overwritten. When Wbox() is called a second time, the window is reduced again. If a window is too small to frame, Wbox() returns a value of -1 and changes nothing.

If you want to put a title in a frame, you can place the title in the top row of the window border by specifying @..SAY for the row but will need to call Wformat() first, see example below.

This function apply for Terminal i/o mode only. It is ignored in GUI mode, since the widget already has own frame.

**Example:**

```
w1 := wopen(10,10, 20,50)
@ 2,2 say ltrim(MaxRow()) + " x " + ltrim(MaxCol()) // 10 x 40
wait
wbox(NIL, "r+/bg")
@ 3,2 say ltrim(MaxRow()) + " x " + ltrim(MaxCol()) // 8 x 38
```

**Example:**

simulate the Wcaption() function

```
w1 := wopen(10,10, 20,50)
wbox(NIL, "r+/bg")           // draw the box

// same as wcaption("Add-on window", NIL, "r+/bg")
wformat()                   // cancel hiding border
@ 0,2 say "Add-on window" color "r+/bg"
wformat(1,1,1,1)           // enable hiding border
```

**Classification:**

add-on library, programming, windowing

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools provides support for the first argument only. Apply for Terminal i/o mode, ignored in GUI mode.

**Related:**

Wopen(), Wselect(), @..BOX, Wcaption(), SET COLOR

# WCAPTION ()

---

**Syntax:**

```
ret = Wcaption ( [expC1], [expN2], [expC3] )
```

**Purpose:**

Write textual caption in the window frame or return current one.

**Arguments:**

<expC1> is an optional character string specifying the caption text to be displayed in the box header. If not specified, the current caption is returned instead.

<expN2> is the window ID# for which this function apply. If not specified, the currently selected sub-window is used.

<expC3> is an optional string with color specification of the frame. Only the first color pair is used. If not specified, the current sub-windows SET COLOR setting is used, if any. Apply for Terminal i/o mode only, ignored otherwise.

**Returns:**

<ret> is a string (or a NIL value) of the currently set caption.

**Description:**

Wcaption() sets the specified text into the windows header frame. In Terminal i/o mode, it assumes that Wbox() is already issued.

**Example:**

```
w1 := wopen(10,10, 20,50)
wbox(NIL, "r+/bg")           // draw the box

wcaption("Add-on window", NIL, "r+/bg")

// the above wcaption(...) is same as manually invoking:
// wformat()                // cancel hiding border
// @ 0,2 say "Add-on window" color "r+/bg"
// wformat(1,1,1,1)         // enable hiding border
```

**Classification:**

add-on library, programming, windowing

**Compatibility:**

Available in FS2-Toolbox only.

**Related:**

Wopen(), Wselect(), Wbox(), Wformat()

# WCENTER ()

---

**Syntax:**

`retN = Wcenter ( [expL1], [expN2] )`

**Purpose:**

Center the specified or current window in the middle of main screen

**Arguments:**

<expL1> is an optional logical value. If the parameter is .T., the function computes the center of the current or by <expN2> specified sub-window and moves it to that position. If the parameter is .F., the function returns a window to the visible screen area. The default value is .F. for Terminal i/o and .T. otherwise.

<expN2> is the window ID# for which this function apply. If not specified, the currently selected sub-window is used.

**Returns:**

<retN> is the ID# number of the currently selected sub-window or -1 if the call fails.

**Description:**

Wcenter() can take a window that has been moved out of visible range and make it fully visible. It also can center the current or specified sub-window in the middle of main screen.

**Example:**

```
w1 := wopen(10,10, 20,50)
wmove(5,5)
wait "press any key to center this window"
wcenter(.T.)
...
// make sure the window is visible after interactive movement
wcenter()
```

**Classification:**

add-on library, programming, windowing

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools provides only support for the first argument.

**Related:**

Wselect(), WMove()

# WCLOSE ()

---

**Syntax:**

```
retN = Wclose ( [expN1] )
```

**Purpose:**

Close the current or specified sub-window open by Wopen() or by MDIopen()

**Arguments:**

<expN1> is the window ID# for which this function apply. If not specified, the currently selected sub-window is closed, except it is the main screen with ID#0.

**Returns:**

<retN> is the ID# number of the currently selected sub-window, or -1 if the call fails.

**Description:**

Wclose() closes the currently selected sub-window and select a sub-window with the highest ID#. Wclose(expN1) closes the specified sub-window and does not change the selection.

The ID# (handle) of the closed sub-window may be re-used by the next Wopen() call. Using <expN1> = 0 or invoking Wclose() without parameter from the main screen is ignored.

Wclose() of MDI sub-window is equivalent to the standard function MDIclose().

Note: the close icon [X] displayed in the sub-window title frame in GUI mode is created by the window manager, not by FlagShip self. An user click on this button (in sub-window) is simply ignored, as opposite to the same button of the whole application which is handled by the InitloQuit() function, see details in the initiomenu.prg source. To close the sub-window programmatically, use this Wclose() function.

**Example:**

```
w1 := wopen(10,10, 20,50)
w2 := wopen(20,20, 25,60)
w3 := wopen( 5,39, 15,60)

w := wselect(w1)
w := wclose()           // closes w1, w3 is selected thereafter
w := wclose(w2)         // closes w2, w3 remain selected
w := wclose()           // closes w3, main screen ID#0 is selected
```

**Classification:**

add-on library, programming, windowing

**Compatibility:**

For MDI mode, the standard MDIclose() function behaves like Wclose() Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools does not support the optional argument and can hence close only the currently selected window.

**Related:** Wopen(), Wselect(), WaClose(), MDIopen(), MDIselect(), MDIclose()

# WCOL ()

---

## Syntax:

**retN = Wcol ( [expL1], [expL2], [expN3] )**

## Purpose:

Return the column number of left frame of the current or specified sub-window.

## Arguments:

<expL1> if this parameter is .T., the returned column is a calculated position corresponding to a column which would be set when calling Wcenter(.T.) - but without moving the window now. If this parameter is .F. or was not specified, Wcol() returns the current position.

<expL2> is an optional pixel specification. If .T., the returned coordinate is in pixel, if .F. the coordinate is col, otherwise the current SET PIXEL status is used.

<expN3> is the window ID# for which this function apply. If not specified, the currently selected sub-window is used, except it is the main screen with ID#0.

## Returns:

<retN> is the column of the left frame (in coordinates or pixel) or -1 if the call fails, i.e. if the sub-window is not open.

## Description:

Wcol(), WlastCol(), Wrow(), WlastRow() functions determine the current window's frame position. You can use them to e.g. move the window back after an user movement or to restore previous values. You also can check the position what the window position would be after Wcenter(.T.) will be called.

Values can be negative. Negative values used in Wmove() place the first column outside of the main screen, if possible.

Note the difference between Wcol() and the standard Col() function: Wcol() acts relatively to the sub-window and returns the window's position within the main screen, whilst Col() reports the current cursor position within the selected sub-window.

## Example:

```
w1 := wopen(10,12,20,50)
@ 2,0 say "Hello"
y := Row()
x := Col()
? y,x           // 2 5
? wrow(), wcol() // 10 12

WmoveUser(.T.)   // or ksetScroll(.T.)
wait = "Move window with cursor keys"
WmoveUser(.F.)   // or ksetScroll(.F.)

nR := wrow()
nC := wcol()
```

```
waclose()
```

```
// ... later: re-open at the user's preferred position  
w1 := wopen(nR,nC,nR+10,nC+40)
```

**Classification:**

add-on library, programming, windowing

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only the first optional argument.

**Related:**

Wopen(), Wrow(), Col(), WlastCol()

# WFCOL ()

---

## Syntax:

**retN** = **WfCol** ( [**expL1**] , [**expL2**] , [**expN3**] )

## Purpose:

Returns the position of the leftmost column of the formatted area of the current or specified sub-window.

## Options:

<**expL1**> is an optional logical value. If .T., <retN> is the absolute value of the difference between Wcol() and WfCol(). Otherwise <retN> is the position of the leftmost column of the formatted area within a sub-window.

<**expL2**> is an optional pixel specification. If .T., the returned coordinate is in pixel, if .F. the coordinate is col, otherwise the current SET PIXEL status is used.

<**expN3**> is the window ID# for which this function apply. If not specified, the currently selected sub-window is used.

## Returns:

<**retN**> is the position of the leftmost column of the formatted area within a window, or the difference between the outer border and the formatted area of a window. In GUI mode, <retN> is always 0.

## Description:

WfCol(), WfLastCol(), WfRow(), WfLastRow() functions are useful for saving current sub-window properties for later use. In order to avoid complex calculations, you can determine the difference between the outer border and the formatted area when you use the optional logical parameter <expL1>. You can then pass this value when you call Wformat() to restore the old value.

When used in Window# 0, the function returns the value for Wboard().

In GUI mode, Wformat() does not change the visible area and WfCol() therefore return 0.

## Example:

```
wopen(10, 10, 20, 70)
wbox()                                // same as wformat(1,1,1,1)

? wfRow(), wfRow(.T.)                 // 11, 1
? wfCol(), wfCol(.T.)                 // 11, 1
? wfLastRow(), wfLastRow(.T.)         // 19, 1
? wfLastCol(), wfLastCol(.T.)         // 69, 1
```

**Classification:** add-on library, programming, windowing

**Compatibility:** Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only the first optional argument.

**Related:** WCol(), Wbox(), Wformat(), WfLastCol(), WfRow(), WfLastRow()



# WFLASTCOL ()

---

**Syntax:**

**retN = WfLastCol ( [expL1], [expL2], [expN3] )**

**Purpose:**

Returns the position of the rightmost column of the formatted area of the current or specified sub-window.

**Options:**

<expL1> is an optional logical value. If .T., <retN> is the absolute value of the difference between WlastCol() and WfLastCol(). Otherwise <retN> is the position of the rightmost column of the formatted area within a sub-window.

<expL2> is an optional pixel specification. If .T., the returned coordinate is in pixel, if .F. the coordinate is col, otherwise the current SET PIXEL status is used.

<expN3> is the window ID# for which this function apply. If not specified, the currently selected sub-window is used.

**Returns:**

<retN> is the position of the rightmost column of the formatted area within a window, or the difference between the outer border and the formatted area of a window. In GUI mode, <retN> is always 0.

**Description:**

WfCol(), WfLastCol(), WfRow(), WfLastRow() functions are useful for saving current sub-window properties for later use. In order to avoid complex calculations, you can determine the difference between the outer border and the formatted area when you use the optional logical parameter <expL1>. You can then pass this value when you call Wformat() to restore the old value.

When used in Window# 0, the function returns the value for Wboard().

In GUI mode, Wformat() does not change the visible area and hence WfLastCol() return 0.

**Example:**

see WfCol()

**Classification:**

add-on library, programming, windowing

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only the first optional argument.

**Related:**

WCol(), WlastCol(), Wformat(), WfRow(), WfLastRow()

# WFLASTROW ()

---

**Syntax:**

**retN** = WfLastRow ( [expL1], [expL2], [expN3] )

**Purpose:**

Returns the position of the bottom row of the formatted area of the current or specified sub-window.

**Options:**

<expL1> is an optional logical value. If .T., <retN> is the absolute value of the difference between WlastRow() and WfLastRow(). Otherwise <retN> is the position of the bottom row of the formatted area within a sub-window.

<expL2> is an optional pixel specification. If .T., the returned coordinate is in pixel, if .F. the coordinate is col, otherwise the current SET PIXEL status is used.

<expN3> is the window ID# for which this function apply. If not specified, the currently selected sub-window is used.

**Returns:**

<retN> is the position of the bottom row of the formatted area within a window, or the difference between the outer border and the formatted area of a window. In GUI mode, <retN> is always 0.

**Description:**

WfCol(), WfLastCol(), WfRow(), WfLastRow() functions are useful for saving current sub-window properties for later use. In order to avoid complex calculations, you can determine the difference between the outer border and the formatted area when you use the optional logical parameter <expL1>. You can then pass this value when you call Wformat() to restore the old value.

When used in Window# 0, the function returns the value for Wboard().

In GUI mode, Wformat() does not change the visible area and hence WfLastRow() return 0.

**Example:**

see WfCol()

**Classification:**

add-on library, programming, windowing

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only the first optional argument.

**Related:**

Wrow(), WfRow(), Wformat(), WfCol(), WfLastCol()

# WFORMAT ()

---

## Syntax:

```
retN = Wformat ( [expN1], [expN2], [expN3],  
                  [expN4], [expL5], [expN6] )
```

## Purpose:

Specifies the usable area within a sub-window. Apply for Terminal i/o mode only, ignored in GUI mode.

## Options:

<expN1>...<expN4> are the top, left, bottom, right coordinates of the accessible sub-window area. When <expNx> is not specified, the default value is 0.

<expL5> is an optional pixel specification. If .T., the coordinates are in pixel, if .F. the coordinates are row/col, otherwise the current SET PIXEL status is used. Apply for GUI mode only, ignored otherwise.

<expN6> is the window ID# for which this function apply. If not specified, the currently selected sub-window is used, except it is the main screen with ID#0.

## Returns:

<retN> is the ID# number of the currently selected sub-window, or -1 if the call fails.

## Description:

This function can be used to reduce the area in which text output occurs compared to the actual area of the sub-window. This allows you to define a margin between the text and the borders of a window where you cannot write. The four parameters define the margin between the old border and the output area.

The sum of the <expN1> and <expN3> values should not exceed the internal height of the sub-window and that the sum of the <expN2> and <expN4> values must not exceed the internal width of the sub-window. In any size window, the output area can be reduced down to one row and column.

Negative values increase the text output area while positive values reduce it. A Wformat() with very large negative values removes all borders.

This function cannot be used with the physical screen (WinID# 0)

To write in the border area, you need to disable Wformat() first, since negative @..SAY values are not allowed. See example below.

## Example:

```
w1 := wopen(10,10, 20,50)  
wbox(NIL, "r+/bg")           // draw the box, set wformat(1,1,1,1)  
  
wformat()                     // cancel hiding border  
@ 0,2 say "Add-on window" color "r+/bg"  
wformat(1,1,1,1)             // hide border anew
```

**Classification:**

add-on library, programming, windowing

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only the first four optional arguments.

**Related:**

Wbox(), Wopen(), Wselect(), WaClose()

# WfRow ()

---

**Syntax:**

**retN = WfRow ( [expL1] , [expL2] , [expN3] )**

**Purpose:**

Returns the position of the top row of the formatted area of the current or specified sub-window.

**Options:**

**<expL1>** is an optional logical value. If .T., <retN> is the absolute value of the difference between Wrow() and WfRow(). Otherwise <retN> is the position of the top row of the formatted area within a sub-window.

**<expL2>** is an optional pixel specification. If .T., the returned coordinate is in pixel, if .F. the coordinate is col, otherwise the current SET PIXEL status is used.

**<expN3>** is the window ID# for which this function apply. If not specified, the currently selected sub-window is used.

**Returns:**

**<retN>** is the position of the top row of the formatted area within a window, or the difference between the outer border and the formatted area of a window. In GUI mode, <retN> is always 0.

**Description:**

WfCol(), WfLastCol(), WfRow(), WfLastRow() functions are useful for saving current sub-window properties for later use. In order to avoid complex calculations, you can determine the difference between the outer border and the formatted area when you use the optional logical parameter <expL1>. You can then pass this value when you call Wformat() to restore the old value.

When used in Window# 0, the function returns the value for Wboard().

In GUI mode, Wformat() does not change the visible area and hence WfRow() return 0.

**Example:**

see WfCol()

**Classification:**

add-on library, programming, windowing

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only the first optional argument.

**Related:**

Wrow(), WlastRow(), WfLastRow(), Wformat(), WfCol(), WfLastCol()

# WLASTCOL ()

---

**Syntax:**

**retN = WlastCol ( [expL1], [expL2], [expN3] )**

**Purpose:**

Return the column number of right frame of the current or specified sub-window.

**Arguments:**

**<expL1>** if this parameter is .T., the returned column is a calculated position corresponding to a column which would be set when calling Wcenter(.T.) - but without moving the window now. If this parameter is .F. or was not specified, WlastCol() returns the current position.

**<expL2>** is an optional pixel specification. If .T., the returned coordinate is in pixel, if .F. the coordinate is col, otherwise the current SET PIXEL status is used.

**<expN3>** is the window ID# for which this function apply. If not specified, the currently selected sub-window is used, except it is the main screen with ID#0.

**Returns:**

**<retN>** is the column of the right frame (in coordinates or pixel) or -1 if the call fails, i.e. if the sub-window is not open.

**Description:**

Wcol(), WlastCol(), Wrow(), WlastRow() functions determine the current window's frame position. You can use them to e.g. move the window back after an user movement or to restore previous values. You also can check the position what the window position would be after Wcenter(.T.) will be called.

Values can be negative. Negative values used in Wmove() place the first column outside of the main screen, if possible.

Note the difference between WlastCol() and WmaxCol() function: WlastCol() acts relatively to the sub-window and returns the window's position within the main screen, whilst WmaxCol() reports the max available column in the selected sub-window.

**Example:**

see Wcol()

**Classification:**

add-on library, programming, windowing

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only the first optional argument.

**Related:**

Wcol(), Wrow(), WlastRow(), Col(), WmaxCol()

# WLASTROW ()

---

**Syntax:**

`retN = WlastRow ( [expL1], [expL2], [expN3] )`

**Purpose:**

Return the row number of bottom frame of the current or specified sub-window.

**Arguments:**

<expL1> if this parameter is .T., the returned row is a calculated position corresponding to a row which would be set when calling Wcenter(.T.) - but without moving the window now. If this parameter is .F. or was not specified, WlastRow() returns the current position.

<expL2> is an optional pixel specification. If .T., the returned coordinate is in pixel, if .F. the coordinate is col, otherwise the current SET PIXEL status is used.

<expN3> is the window ID# for which this function apply. If not specified, the currently selected sub-window is used, except it is the main screen with ID#0.

**Returns:**

<retN> is the row of the bottom frame (in coordinates or pixel) or -1 if the call fails, i.e. if the sub-window is not open.

**Description:**

Wcol(), WlastCol(), Wrow(), WlastRow() functions determine the current window's frame position. You can use them to e.g. move the window back after an user movement or to restore previous values. You also can check the position what the window position would be after Wcenter(.T.) will be called.

Values can be negative. Negative values used in Wmove() place the first column outside of the main screen, if possible.

Note the difference between WlastRow() and the WmaxRow() function: WlastRow() acts relatively to the sub-window and returns the window's position within the main screen, whilst WmaxRow() reports the max available row in the selected sub-window.

**Example:**

see Wcol()

**Classification:**

add-on library, programming, windowing

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only the first optional argument.

**Related:**

Wrow(), Wcol(), WlastCol(), Row(), WMaxRow()

# WMAXCOL ()

---

## Syntax:

**retN = WmaxCol ( [expL1], [expL2], [expN3] )**

## Purpose:

Return the max available column in the current or specified sub- window or of the main screen. WmaxCol() without parameters is equivalent to the standard function MaxCol(), see section FUN.

## Options:

**<expL1>** When .F. is passed, or when no parameter is specified, the function returns the highest column number of the selected window and is hence equivalent to the standard function MaxCol(). When .T. is passed, it returns the highest column number for the main screen and hence WmaxCol(.T.) is equivalent to WmaxCol(.,0) or to Max\_Col() or to MaxCol(.T.) from NT2/CA3. Note: the standard FlagShip function MaxCol(.T.) has another meaning, returning the coordinate in pixel.

**<expL2>** is an optional pixel specification. If .T., the returned coordinate is in pixel, if .F. the coordinate is col, otherwise the current SET PIXEL status is used.

**<expN3>** is the window ID# for which this function apply. If not specified, the currently selected sub-window is used.

## Returns:

**<retN>** is the column coordinate of the current or specified window.

## Description:

WmaxCol() is very similar to the standard MaxCol() function and differs to it only with the passed arguments, if such used.

In GUI mode, Wformat() does not change the visible area and hence WmaxCol() return the same value after Wformat(...).

## Example:

```
w1 := wopen(10,10, 20,50)
? WmaxCol(), MaxCol()           // 41 41
? WmaxCol(.T.), WMaxCol(.,0)    // 79 79
wselect(0)
? WmaxCol(), MaxCol()           // 79 79
wselect(w1)
wformat(1,1,1,1)
? WmaxCol(), MaxCol()           // 39 30 (or 41 41 in GUI mode)
```

## Classification:

add-on library, programming, windowing

## Compatibility:

Available in FS2-Toolbox, not available in NT2/CA3 tools where the overlaid MaxCol() in Extended Drivers accept the optional <expL1> parameter. Hence WmaxCol(.T.) from FS2 behaves same as MaxCol(.T.) from CA3. WmaxCol() w/o



parameters is equivalent to the standard function MaxCol() which behaves same in FlagShip, Clipper, FS2 and CA3.

For cross-compatible applications, you may use

```
#ifndef FlagShip5
# translate wmaxcol([<p>]) => maxcol(<p>)
# translate wmaxrow([<p>]) => maxrow(<p>)
#endif
```

***Related:***

MaxCol(), WmaxRow(), Wopen(), Wselect(), Wformat(), NumCol()

# WMAXROW ()

---

## Syntax:

**retN = WmaxRow ( [expL1] , [expL2] , [expN3] )**

## Purpose:

Return the max available row in the current or specified sub- window or of the main screen. WmaxRow() without parameters is equivalent to the standard function MaxRow(), see section FUN.

## Options:

**<expL1>** When .F. is passed, or when no parameter is specified, the function returns the highest row number of the selected window and is hence equivalent to the standard function MaxRow(). When .T. is passed, it returns the highest row number for the main screen and hence WmaxRow(.T.) is equivalent to WmaxRow(.,0) or to Max\_Row() or to MaxRow(.T.) from NT2/CA3. Note: the standard FlagShip function MaxRow(.T.) has another meaning, returning the coordinate in pixel.

**<expL2>** is an optional pixel specification. If .T., the returned coordinate is in pixel, if .F. the coordinate is col, otherwise the current SET PIXEL status is used.

**<expN3>** is the window ID# for which this function apply. If not specified, the currently selected sub-window is used.

## Returns:

**<retN>** is the row coordinate of the current or specified window.

## Description:

WmaxRow() is very similar to the standard MaxRow() function and differs to it only with the passed arguments, if such used.

In GUI mode, Wformat() does not change the visible area and hence WmaxRow() return the same value after Wformat(...).

## Example:

```
w1 := wopen(10,10, 20,50)
? WmaxRow(), MaxRow()           // 11 11
? WmaxRow(.T.), WMaxRow(.,0)    // 25 25
wselect(0)
? WmaxRow(), MaxRow()           // 25 25
wselect(w1)
wformat(1,1,1,1)
? WmaxRow(), MaxRow()           // 9 9 (or 11 11 in GUI mode)
```

## Classification:

add-on library, programming, windowing

## Compatibility:

Available in FS2-Toolbox, not available in NT2/CA3 tools where the overlaid MaxRow() in Extended Drivers accept the optional <expL1> parameter. Hence WmaxRow(.T.) from FS2 behaves same as MaxRow(.T.) from CA3. WmaxRow()

w/o parameters is equivalent to the standard function MaxRow() which behaves same in FlagShip, Clipper, FS2 and CA3.

For cross-compatible applications, you may use

```
#ifndef FlagShip5
# translate wmaxcol([<p>]) => maxcol(<p>)
# translate wmaxrow([<p>]) => maxrow(<p>)
#endif
```

***Related:***

MaxRow(), WmaxCol(), Wopen(), Wselect(), Wformat()

# WMIDDLE ()

---

**Syntax:**

**retN = Wmiddle ( [expN1] )**

**Purpose:**

Center the current sub-window, same as Wcenter(.T., [expN1]).

**Options:**

<expN1> is the window ID# for which this function apply. If not specified, the currently selected sub-window is used, except it is the main screen with ID#0.

**Returns:**

<retN> is the ID# number of the currently selected sub-window, or -1 if the call fails.

**Description:**

Wmiddle() behaves same as Wcenter(.T.)

**Example:**

see Wcenter()

**Classification:**

add-on library, programming, windowing

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools where undocumented.

**Related:**

Wcenter()

# WMODE ()

---

**Syntax:**

```
retN = Wmode ( [expL1] , [expL2] , [expL3] , [expL4] ,  
               [expL5] , [expN6] )
```

**Purpose:**

Turns the screen border overstep mode on or off. Apply for Terminal i/o mode only, ignored in GUI mode.

**Options:**

<expL1> if .T., allow overstep the top border

<expL2> if .T., allow overstep the left border

<expL3> if .T., allow overstep the right border

<expL4> if .T., allow overstep the bottom border

<expL5> is an optional pixel specification. If .T., the returned coordinate is in pixel, if .F. the coordinate is col, otherwise the current SET PIXEL status is used.

<expN6> is the window ID# for which this function apply. If not specified, the currently selected sub-window is used, except it is the main screen with ID#0.

**Returns:**

<retN> is 0 on success (and in GUI mode), or -1 if the call fails.

**Description:**

With Wmode() you can define whether each screen page can be moved, interactively or with Wmove(), beyond the edge of each side of the main screen or beyond the area defined by Wboard(). The Wmode() setting apply for all open sub-windows.

**Example:**

Allow that the sub-windows can only be moved over the top or bottom of the screen:

```
WMODE(.T., .F., .T., .F.)
```

**Classification:**

add-on library, programming, windowing

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only the first four optional arguments.

**Related:**

Wopen(), Wselect(), Wmove(), Wboard()

# WMOVE ()

---

**Syntax:**

`retN = Wmove ( expN1, expN2, [expL3], [expN4] )`

**Purpose:**

Moves the current or specified sub-window.

**Parameters:**

<expN1> is a row coordinate of the upper left edge specifying the new window position

<expN2> is a column coordinate of the upper left edge specifying the new window position

<expN1> and <expN2> can have negative values to move the window out of the main screen. Supported only in GUI mode for non MDI windows.

**Options:**

<expL3> is an optional pixel specification. If .T., the <expN1> and <expN2> coordinate is in pixel, if .F. these coordinates are row/col, otherwise the current SET PIXEL status is used.

<expN4> is the window ID# for which this function apply. If not specified, the currently selected sub-window is used, except it is the main screen with ID#0.

**Returns:**

<retN> is the ID# number of the currently selected sub-window, or -1 if the call fails.

**Description:**

Wmove() allows you to move the currently selected window to a particular position or within the area defined by Wboard().

Note: WsetMove() does not affect the operation of Wmove(), but Wmode() affects the operation of Wmove().

**Example:**

```
w1 := wopen(2,2, 19,15)
w2 := wopen(10,10, 20,50)
wmove(5,15)           // move win#2 from 10,10 to 5,15
wmove(wrow()-1, wcol()-1, w1) // move win#1 one row up & col left
```

**Classification:**

add-on library, programming, windowing

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only the first two arguments.

**Related:**

Wopen(), Wselect(), WaClose(), WmoveDown(), WmoveUp(), WmoveLeft(), WmoveRight()

# WMOVEDOWN ()

---

**Syntax:**

**NIL = WmoveDown ( [expN1] )**

**Purpose:**

Move sub-window down number of rows set by Wstep().

**Options:**

<expN1> is the window ID# for which this function apply. If not specified, the currently selected sub-window is used, except it is the main screen with ID#0.

**Description:**

Moves the current or specified sub-window down one step specified by Wstep(). The default are 2 rows. WmodeDown() is equivalent to user movement by one cursor-down key press.

**Example:**

```
w1 := wopen(10,10, 15,30)
wmoveDown()
```

**Classification:**

add-on library, programming, windowing

**Compatibility:**

Available in FS2-Toolbox, not available in NT2/CA3 tools.

**Related:**

Wmove(), Wstep(), WmoveUp(), WmoveLeft(), WmoveRight(), WmoveUser()

# WMOVELEFT ()

---

**Syntax:**

**NIL = WmoveLeft ( [expN1] )**

**Purpose:**

Move sub-window left number of columns set by Wstep().

**Options:**

<expN1> is the window ID# for which this function apply. If not specified, the currently selected sub-window is used, except it is the main screen with ID#0.

**Description:**

Moves the current or specified sub-window left one step specified by Wstep(). The default are 6 cols. WmodeDown() is equivalent to user movement by one cursor-left key press.

**Example:**

```
w1 := wopen(10,10, 15,30)
wmoveLeft()
```

**Classification:**

add-on library, programming, windowing

**Compatibility:**

Available in FS2-Toolbox, not available in NT2/CA3 tools.

**Related:**

Wmove(), Wstep(), WmoveDown(), WmoveUp(), WmoveRight(), WmoveUser()



# WMOVERIGHT ()

---

**Syntax:**

`NIL = WmoveRight ( [expN1] )`

**Purpose:**

Move sub-window right number of columns set by Wstep().

**Options:**

<expN1> is the window ID# for which this function apply. If not specified, the currently selected sub-window is used, except it is the main screen with ID#0.

**Description:**

Moves the current or specified sub-window right one step specified by Wstep(). The default are 6 cols. WmodeDown() is equivalent to user movement by one cursor-right key press.

**Example:**

```
w1 := wopen(10,10, 15,30)
wmoveRight()
```

**Classification:**

add-on library, programming, windowing

**Compatibility:**

Available in FS2-Toolbox, not available in NT2/CA3 tools.

**Related:**

Wmove(), Wstep(), WmoveDown(), WmoveUp(), WmoveLeft(), WmoveUser()

# WMOVEUP ()

---

**Syntax:**

**NIL = WmoveDown ( [expN1] )**

**Purpose:**

Move sub-window upwards number of rows set by Wstep().

**Options:**

<expN1> is the window ID# for which this function apply. If not specified, the currently selected sub-window is used, except it is the main screen with ID#0.

**Description:**

Moves the current or specified sub-window up one step specified by Wstep(). The default are 2 rows. WmodeDown() is equivalent to user movement by one cursor-up key press.

**Example:**

```
w1 := wopen(10,10, 15,30)
wmoveUp()
```

**Classification:**

add-on library, programming, windowing

**Compatibility:**

Available in FS2-Toolbox, not available in NT2/CA3 tools.

**Related:**

Wmove(), Wstep(), WmoveDown(), WmoveLeft(), WmoveRight(), WmoveUser()

# WMOVEUSER () KSETSCROLL ()

---

**Syntax:**

```
retL = WmoveUser ( [expL1], [expA2] )
```

**Syntax 2:**

```
retL = KsetScroll ( [expL1], [expA2] )
```

**Purpose:**

Enable/disable window movement by cursor keys.

**Options:**

<expL1> is an optional logical value. If .T., the the user movement by cursor keys is enabled, when .F. it is disabled. The default is .F. If the parameter is not specified, WmoveUser() returns the current setting.

<expA2> is an optional array with 1 to 4 numeric values specifying the Inkey code for left, right, up and down movement (in that order). If the array element is not numeric, default key value is used. Zero value disables movement in that direction. The default setting is {K\_LEFT, K\_RIGHT, K\_UP, K\_DOWN}, see also .../include/inkey.fh file. This <expA2> parameter is considered only with <expL1> == .T. and ignored otherwise.

**Returns:**

<retL> is the current status at the time of entering the function.

**Description:**

This function turns the interactive movement mode for sub-windows on or off, or returns the current status only. The sub-window movement is possible only if WsetMove() is .T. which is the default.

**Example:**

```
w1 := wopen(10,12,20,50)
wCaption("First window")
@ 2,0 say "Hello"

wmoveUser(.T.)           // or KsetScroll(.T.)
wait "Move this window with Cursor keys"

wmoveUser(.T., {K_ALT_LEFT, K_ALT_RIGHT, K_ALT_UP, K_ALT_DOWN} )
wait "Move this window with Alt + Cursor keys"

wmoveUser(.F.)
wait "user-movement disabled now..."
```

**Classification:**

add-on library, programming, windowing

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools KsetScroll(). The 2nd parameter of KsetScroll() is available in FlagShip only. As opposed to NT2/CA3, FlagShip's version is independent on locked SCROLL key.

**Related:**

Wopen(), WsetMove(), Wmove(), WmoveDown(), WmoveLeft() etc.

# WNUM ( )

---

**Syntax:**

**retN = Wnum ( )**

**Purpose:**

Determines the highest window ID#.

**Returns:**

<retN> is the highest ID# number of all open sub-windows, or 0 if no sub-windows are open.

**Description:**

By using Wnum() you can iterate a loop over the range of all open sub-windows. Note that the ID# number of sub-windows may start with any positive value > 0 and that closed windows does not re-arrange the window ID# numbers, so the range of 1..<retN> may contain unused ID# numbers.

**Example:**

Simulate the WaClose() function:

```
w1 := wopen(3, 3, 10, 12)
w2 := wopen(10, 10, 20, 70)
w3 := wopen(5, 5, 22, 60)

for iLoop := 1 to wnum()
  if wselect(iLoop) > 0
    wclose()
  endif
next
```

**Classification:**

add-on library, programming, windowing

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

Wopen(), Wselect(), Wclose(), WaClose()

# WOPEN ()

---

## Syntax:

```
retN = Wopen ( expN1, expN2, expN3, expN4, [expL5],  
               [expC6], [expN7], [expL8], [expC9] )
```

## Purpose:

Open a new sub-window.

## Arguments:

<expN1>...<expN4> are the top, left, bottom, right coordinates of the new sub-window, either in row/col or in pixel. Accepted values are between 0..MaxRow() and 0..MaxCol()

## Options:

<expL5> is an optional logical value, specifying whether the area of <expN1>...<expN4> in the main screen should be erased. When .T., the main screen area is erased, otherwise it is not. Apply for Terminal i/o mode, ignored in GUI mode.

<expC6> is an optional string, specifying the sub-window caption. If not empty(), Wopen() calls Wcaption(<expC6>). If not specified, the sub-window caption in GUI mode is "Window #nn" where nn is the window ID# returned by Wopen(). In Terminal i/o mode, when <expC6> is not specified or is an empty string, no action is taken; otherwise the caption text is displayed within a border drawn with \_aGlobSetting[GSET\_T\_C\_AT\_TO\_SINGLE] or <expC9> frame.

<expN7> is the sub-window screen appearance, considered in GUI mode and ignored otherwise. The possible values are 0, 1 or 2:

- 0: (the default) determine the mode automatically: use mode 2 for MDI applications, otherwise use mode 1.
- 1: create semi-modal sub-window. Applicable for SDI and MDI mode. The menu of the application is selectable, and the GUI debugger can be used. The behavior is very similar to sub-windows in Terminal i/o mode. The sub-window is managed by the window manager, so it may be covered by other windows, including the application window self. Temporarily covered sub-window by other windows can be selected and become visible via Panel of the window manager or programmable via Wselect().
- 2: create additional MDI sub-window, same as with the standard MDIopen() function. Accepted in MDI mode only, i.e. when the application was compiled with the -mdi switch.

<expL8> is an optional pixel specification. If .T., the coordinates are in pixel, if .F. the coordinates are row/col, otherwise the current SET PIXEL status is used. Apply for GUI mode only, ignored otherwise.

<expC9> is an optional string specifying the border frame drawn around the sub-window. Considered in Terminal i/o mode only. If not empty, Wopen() invokes Wbox(<expC9>).

**Returns:**

<retN> is the ID# number of the new and selected sub-window, or -1 if the call fails. You will need to store the return value to a variable, to be able to access/select it later by Wselect(idNum) or to close the sub-window via Wclose(idNum).

Warnings and errors are reported in Developer's mode, i.e. when FS\_SET("devel",.T.) is set. Otherwise check the <retN> for value <= 0.

**Description:**

Wopen() creates and opens a new sub-window. If successful, it returns a number (handle) for this window and select it. If the coordinates are invalid, or the call fails for other reason, Wopen() returns -1 and the window that was active at the time of the function call is still active. There is no limit of used sub-windows in FlagShip.

A sub-window is a virtual screen in terminal i/o, or is a separate widget (control) in GUI. From programmer's view, it differs from the main screen only in size. After you open new sub-window, all screen output is redirected to the window and this sub-window receives input focus. The underlying screen area of the main screen is saved automatically when a new window is created. This applies equally to any area of the main screen that becomes overlapped by the movement of a sub-window.

The sub-window behaves same as the main screen. All the standard settings, including SET PIXEL ON|OFF apply also for the selected sub-window. But local row/column status and SET COLOR settings apply for the currently selected sub-window only.

If a shadow has been selected using WsetShadow(), the window screen area is cleared when you open the window (in Terminal i/o mode). When you specify window caption via <expC6>, a box is drawn around the window and the caption is displayed in the box top line, same as calling Wbox() and Wcaption(<expC6>) explicitly.

You can retrieve the active window ID# by calling Wselect() with no parameters. In GUI, the active window is marked by [\*] sign in the header and with "checked" mark in the Menu->Windows->Sub-Windows list. You may select and view an inactive window (read-only) by click on Menu->Windows->Sub-Windows (user modifiable), see also initimenu.prg for details.

Note: the close icon [X] displayed in the sub-window title frame in GUI mode is created by the window manager, not by FlagShip self. An user click on this button (in sub-window) is simply ignored, as opposite to the same button of the whole application which is handled by the InitloQuit() function, see details in the initimenu.prg source. To close the sub-window programmatically, use Wclose().

**Tuning:**

In GUI mode, Wopen(), Wclose() and Wselect() updates the main menu (oTopBar object) Windows -> Subwindows corresponding to the selected sub-window. This update may however cause short flickering of the main window; you may disable the update by setting

\_aGlobSetting[GSET\_G\_L\_SUBWIN\_UPD\_MENU] := .F. // default is .T.  
or temporary in Wselect() by the 3rd parameter.

**Example:**

```
w1 := wopen(10,10, 20,50)
? "hello first sub-window!"
@ 3,5 SAY "The ID# of this window is",ltrim(w1)
@ 4,5 SAY "MaxRow() * MaxCol() =" + ltrim(MaxRow()) + " x " +
        ltrim(MaxCol())

set color to "w/b,gr+/b"
w2 := wopen(5,5, 15,45, .T., "Second")
? "hello second sub-window!"
@ 3,5 SAY "The ID# of this window is",ltrim(w2)
@ 4,5 SAY "MaxRow() * MaxCol() =" + ltrim(MaxRow()) + " x " +
        ltrim(MaxCol())

set color to
wselect(0)
? "hello on main screen"
@ 3,5 SAY "MaxRow() * MaxCol() =" + ltrim(MaxRow()) + " x " +
        ltrim(MaxCol())

wselect(w1)
? "continuing text in first window"
wait "press any key to close this window..."
wclose()
// highest sub-window = w2 is selected now
? "press any key to move this window"
wait "to 10,15 ..."
wmove(10,15)
wait "any key to close all sub-windows..."
waclose()
```

**Classification:**

add-on library, programming, windowing

**Compatibility:**

For MDI mode, the standard MDIopen() function behaves like Wopen()

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only the first five arguments.

**Related:**

Wselect(), Wclose(), WaClose(), MDIopen(), MDIselect(), MDIclose()



# WROW ()

---

**Syntax:**

**retN = Wrow ( [expL1] , [expL2] , [expN3] )**

**Purpose:**

Return the row number of upper frame of the current or specified sub-window.

**Arguments:**

<expL1> if this parameter is .T., the returned row is a calculated position corresponding to a row which would be set when calling Wcenter(.T.) - but without moving the window now. If this parameter is .F. or was not specified, Wrow() returns the current position.

<expL2> is an optional pixel specification. If .T., the returned coordinate is in pixel, if .F. the coordinate is row, otherwise the current SET PIXEL status is used.

<expN3> is the window ID# for which this function apply. If not specified, the currently selected sub-window is used, except it is the main screen with ID#0.

**Returns:**

<retN> is the row of the upper frame (in coordinates or pixel) or -1 if the call fails, i.e. if the sub-window is not open.

**Description:**

Wcol(), WlastCol(), Wrow(), WlastRow() functions determine the current window's frame position. You can use them to e.g. move the window back after an user movement or to restore previous values. You also can check the position what the window position would be after Wcenter(.T.) will be called.

Values can be negative. Negative values used in Wmove() place the first column outside of the main screen, if possible.

Note the difference between Wrow() and the standard Row() function: Wrow() acts relatively to the sub-window and returns the window's position within the main screen, whilst Row() reports the current cursor position within the selected sub-window.

**Example:**

see Wcol()

**Classification:**

add-on library, programming, windowing

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only the first optional argument.

**Related:**

Wopen(), Wcol(), Row(), WlastRow()

# WSELECT ()

---

**Syntax:**

`retN = Wselect ( [expN1], [expL2], [expL3] )`

**Purpose:**

Activates one of the open sub-windows or the main screen.

**Options:**

<expN1> is the window ID# that is selected. The ID# is returned from Wopen(). Specifying 0 (zero) selects the main screen. When <expN1> is not specified, Wselect() only returns the ID# of the currently selected sub-window.

<expL2> is an optional logical value, specifying if the newly selected sub-window should receive input focus. The default is .T.

<expL3> is an optional logical value, controlling the update of main menu sub-item Windows -> Subwindows. If not specified or is .T., the main menu is updated to correspond selected sub-window. This update may however cause short flickering of the window; you may disable it by passing .F. value to <expL3>. Considered in GUI mode only, ignored otherwise. This parameter overrides the default menu update setting \_aGlobSetting[GSET\_G\_L\_SUBWIN\_UPD\_MENU], see Tuning in Wopen()

**Returns:**

<retN> is the ID# number of the currently selected sub-window, or -1 if the call fails.

**Description:**

Wselect() allows you to take an already open but inactive window and reactivate it. Window 0 corresponds to the main window. By omitting the first parameter you can determine the ID# (handle) of the window currently selected.

When the ID#0 (main screen) is selected, it covers other windows in textual i/o mode.

Wselect(...) for MDI sub-windows is equivalent to the standard function MDIselect(...). In MDI mode, you may align the windows using the "Menu->Windows" selection.

**Example:**

See Wopen()

**Classification:** add-on library, programming, windowing

**Compatibility:** For MDI mode, the standard MDIselect() function behaves like Wselect().

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only the first optional argument.

**Related:** Wopen(), Wclose(), WaClose(), MDIopen(), MDIselect(), MDIclose()

# WSETMOVE ()

---

**Syntax:**

**retL = WsetMove ( [expL1] )**

**Purpose:**

Turns the interactive (user) movement mode on or off.

**Options:**

<expL1> optional logical value specifying whether a window can be moved interactively (.T.) or not (.F.) with the mouse or by cursor keys. The default value (.T.) allows the window to be moved.

**Returns:**

<retL> is the current movement status at the time of invoking this function.

**Description:**

This function allows you to prevent (or permit) a user to interactively move a window with the mouse (in GUI mode only) or with cursor keys. With active WsetMove(), the user can always move the sub-window by mouse. To move it by cursor keys, also WmoveUser() or KsetScroll() must be active, which is (.F.) by default. By invoking WsetMove(.F.) neither user's movement of the sub-window by mouse nor by cursor keys is allowed, regardless the KsetScroll() setting.

If the <expL1> parameter is not specified, the function returns the current status of the switch without changing the current status.

The WsetMove() setting does not affect the Wmove() function; it only affects the interactive setting. The Wmode() setting determines if a sub-window can be moved under a border of an area set by Wboard().

**Example:**

```
w1 := wopen(10,12,20,50)
wmoveUser(.T.)           // enable movement by cursor
wait = "Move window with Cursor keys" + ;
      if(AppIoMode() == "G", " or by mouse", "")
? "any sub-window movement is disabled now"
wsetMove(.F.)
wmoveUser(.F.)           // disable movement by cursor
```

**Classification:**

add-on library, programming, windowing

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

Wmove(), Wmode(), WmoveUser()

# WSETSHADOW ()

---

## Syntax:

**retNC = WsetShadow ( [expNC1], [exp23] )**

## Purpose:

Sets the window shadow colors. Apply for Terminal i/o mode only, ignored otherwise.

## Options:

<expNC1> is an optional color specification, see SET COLOR TO. Only the first color pair is used. If specified numeric -1, turns the current shadowing off. If not specified, WsetShadow() returns the current color setting.

<expN2> is the window ID# for which this function apply. If not specified, the currently selected sub-window is used, except it is the main screen with ID#0.

## Returns:

<retNC> is either numeric -1 when no shadow color has been set, or a string containing the current color setting at the time of invoking this function.

## Description:

WsetShadow() sets individual color attributes for shadows of every sub-window, which are displayed below and to the right of each sub-window frame. The setting then applies to all subsequently opened sub-windows.

Apply for terminal i/o mode only, ignored in GUI.

## Example:

Fill the sub-window with CHR(176) and set shadow:

```
SetClearb("#")
CLEAR                // Fill screen with CHR(176)
wsetShadow("N/R")    // Red shadows
wopen(1, 1, 10, 50)
wbox()

wsetShadow(-1)       // Switch shadow display off
wopen(15, 1, 20, 50)
wbox()

wmoveUser(.T.)       // or ksetScroll(.T.)
wait "move the window by cursor key"
wmoveUser(.F.)       // or ksetScroll(.F.)

wait "done..."
waclose()
```

**Classification:** add-on library, programming, windowing

**Compatibility:** Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only the first optional argument.

**Related:** Wopen(), Wbox(), Wformat()

# WSTEP ( )

---

**Syntax 1:**

```
retN = Wstep ( expN1, expL2 )
```

**Syntax 2:**

```
retA = Wstep ( )
```

**Purpose:**

Set or retrieve the step width of interactive window movement.

**Options:**

<expN1> is the number of rows the window steps when moved interactively by cursor keys. Accepted values are 1 to 6. The default value is 2.

<expN2> is the number of columns the window steps when moved interactively by cursor keys. Accepted values are 1 to 20. The default value is 5.

**Returns:**

<retN> is 0 on success, or -1 if the call fails.

<retA> is 1-dimensional array with two numeric elements reporting the current Wstep() setting: retA[1] the row steps, retA[2] the column steps.

**Description:**

Wstep() allows you to set (or get) the number of steps a window moves with each cursor key stroke.

**Classification:**

add-on library, programming, windowing

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only the syntax 1 and allow the use of Wstep() only when all windows are closed. FS2 accept it at any time.

**Related:**

Wopen(), WmoveUser()

# Video Functions and Extended Drivers

---

## Introduction

The Video functions and Extended Driver in CA3 Tools deals either directly or indirectly with the screen buffer, video adapters, keyboard or BIOS. In FlagShip and in the FS2 Toolbox which works in multiuser environment, such a direct hardware manipulation is not possible. Therefore only a part of these CA3 Tools functions can be supported here in fully. The unsupported functions are available in source code (see fs2\_udfs.prg) and return a standard value to provide backward compatibility to "old" Clipper sources.

### Index of FS2 Video Functions and Extended Drivers

CLEAR_EOL()	Clears to the end of line using FS2 clear char and color
CLEARWIN()	Clears a screen area using FS2 clear character and color
CLEOL()	Clears from the cursor position to the end of line
CLWIN()	Clears a screen area
COL()	Same as standard COL() function, see section FUN
COLORTON()	Converts color string pair into numeric value
DSETTYPE()	Determines the size of the keyboard buffer (SET TYPEAHEAD)
ENHANCED()	Selects the enhanced color value for SET COLOR TO output
FILESCREEN()	Reads screen content from a file
GETCLEARA()	Retrieves the current color for the clearing functions
GETCLEARATTR()	) Retrieves the current color for the clearing functions
GETCLEARB()	Retrieves the default character for the clearing functions
GETCLEARBYTE()	) Retrieves the default character for the clearing functions
INKEYTRAP()	Same as standard INKEYTRAP() function, see section FUN
KEYSEND()	Simulates keyboard buffer input
* MAXCOL()	Replaced by WMAXCOL() function, see FS2:Windowing
* MAXROW()	Replaced by WMAXROW() function, see FS2:Windowing
NUMCOL()	Restores the number of available screen columns
NTOCOLOR()	Converts a numeric value into a color value
ROW()	Same as standard ROW() function, see section FUN
SCREENFILE()	Writes screen content to a file
SCREENSIZE()	Retrieves the size of screen image to be saved
SETCLEARA()	Changes the default color attribute for screen clear
SETCLEARATTR()	) Changes the default color attribute for screen clear
SETCLEARB()	Changes the default character for screen clear
SETCLEARBYTE()	) Changes the default character for screen clear
SETCURSOR()	Same as standard SETCURSOR() function, see section FUN
STANDARD()	Selects the standard color value for SET COLOR TO output
TRAPANYKEY()	Calls a procedure with any keyboard input
UNSELECTED()	Selects the unselected color value for SET COLOR TO output

User modifiable interface to CA3 Tools (available and modifiable in fs2\_udfs.prg) returning standard value or using an emulation:

CGA40()	Switches to 40-column mode (color or monochrome)	.F.
CGA80()	Switches to 80-column mode (color or monochrome)	.F.
CHARPIX()	Returns the number of pixel lines per character	1
CHARWIN()	Exchanges particular characters in a screen area	""
CLEARLOW()	Deletes a screen area from the outside in with a delay	""
COLORREPL()	Exchanges particular screen attributes	""
COLORWIN()	Exchanges particular attributes in a screen area	""
DSETKBIOS()	Turns the extended keyboard mode on or off through BIOS	.T.
DSETNOLINE()	Ignores the next line feed sent to the screen	.F.
DSETQFILE()	Creates a protocol file when the program ends normally	.F.
DSETWINDEB()	This function is no longer supported in CA3	NIL
DSETWINDOW()	Reroutes external functions and programs to a window	.F.
EGA43()	Switches to the 43-line EGA mode	.F.
EGAPALETTE()	Changes EGA palette colors	""
FIRSTCOL()	Sets the first visible column of a virtual screen	0
FIRSTROW()	Sets the first visible line of a virtual screen	0
FONTLOAD()	Loads EGA/VGA fonts from another file	-2
FONTRESET()	Resets all font and palette changes to the ROM defaults	.F.
FONTROTATE()	Rotates and mirrors images within a font string	""
FONTSELECT()	Determines font areas for normal/high-intensity output	-2
GETBOXGROW()	Gets the time delay with which boxes are opened	0
GETCURSOR()	Determines the setting for the cursor form	1
GETFONT()	Retrieves the current font table from video card	""
GETKXLAT()	Determines the current key code table	0
GETKXTAB()	Retrieves the entire key code table	""
GETLINES()	Number of lines after which screen display pauses	0
GETMODE()	Uses the current screen mode as a function name	"CGA80"
GETPAGE()	Determines the current screen page	1
GETPBIOS()	Determines if printing is through DOS or the BIOS	0
GETPXLAT()	Retrieves the current printer table	""
GETSCRMODE()	Determines the number of the active video mode	3
GETSCRSTR()	Queries screen output that was redirected by SETSCRSTR()	""
GETTAB()	Retrieves tab values for screen output	7
GETVGAPAL()	Determines the palette settings on a VGA card	-1
INPUTMODE()	Previously active or currently active input mode	0
INVERTATTR()	Inverts the foreground and background of an attribute	0
INVERTWIN()	Inverts all attributes in an area of the screen	""
ISCGA()	Tests for presence of a CGA card	.T.
ISEGA()	Tests for presence of an EGA card	.T.
ISHERCULES()	Tests for presence of a HERCULES card	.T.
ISMCGA()	Tests for presence of an MCGA card	.T.
ISMONO()	Tests for presence of a monochrome card	.T.
ISPGA()	Tests for presence of a PGA card	.T.
ISVGA()	Tests for presence of a VGA card	.T.

KEYREAD()	Reads already processed keyboard buffer input from BIOS	""
MAXFONT()	Determines the number of available fonts of video card	1
MAXPAGE()	Determines the number of available screen pages	0
MONISWITCH()	Switches between monochrome and color screen	.F.
MONOCHROME()	Switches to the monochrome mode	.F.
PAGECOPY()	Copies one screen page to another	.F.
PRINTERROR()	Returns the error code for the last printer output	0
RESTCURSOR()	Restores a saved cursor position and form	""
SAVECURSOR()	Saves current cursor position and form	0
SAYDOWN()	Displays screen output downward and vertically	""
SAYMOVEIN()	Displays screen output with a "move in" effect	""
SAYSCREEN()	Output to the screen without changing the attribute	(emul)
SAYSPREAD()	Displays screen output with "spread" effect	(emul)
SCREENATTR()	Determines the attribute at a particular position	0
SCREENMARK()	Searches for a string and marks it with an attribute	.F.
SCREENMIX()	Mixes characters and attributes of a screen	""
SCREENSTR()	Reads a string, including attributes, from the screen	""
SETBELL()	Sets the tone frequency and duration for CHR(7)	""
SETBOXGROW()	Opens boxes with a time delay	""
SETFONT()	Loads the font directly out of a string	-1
SETKXLAT()	Redefines key codes or lock keys	.F.
SETKXTAB()	Installs key tables	.F.
SETLINES()	Number of lines after which screen display pauses	""
SETMAXCOL()	Sets the number of columns for a virtual screen	.F.
SETMAXROW()	Sets the number of lines for a virtual screen	.F.
SETPAGE()	Selects a new screen page	.F.
SETPBIOS()	Redirects print output to BIOS or DOS	.T.
SETPXLAT()	Establishes translation tables for printer output	.F.
SETQNAME()	Changes the file and path name for the QUIT file	.F.
SETRC()	Sets line and column for the cursor	""
SETSCRMODE()	Establishes a new video mode	.F.
SETSCRSTR()	Redirects screen output into a string	.F.
SETTAB()	Sets the tab widths for screen outputs	.F.
STRSCREEN()	Displays a string with attributes on the screen	(emul)
TRAPINPUT()	Allows supervision of input commands	.F.
TRAPSHIFT()	Calls a procedure that depends on switching keys	.F.
UNTEXTWIN()	Replaces an area of chars from a region of the screen	""
VGA28()	Switches to 28-line VGA mode	.F.
VGA50()	Switches to 50-line VGA mode.	.F.
VGAPALETTE()	Changes VGA palette colors	.F.
VIDEOINIT()	Reinitializes a video system after a RUN	(emul)
VIDEOSETUP()	Queries video mode at system start	0
VIDEOTYPE()	Returns bit-coded information about available video modes	2



# CLEAR\_EOL ()

---

## Syntax:

```
retC = ClearEol ( [expN1], [expN2], [expC3],  
                  [expC4], [expC5] )
```

## Purpose:

Clears from the cursor position to the end of line, considering the clear byte and clear color attribute set by SetClearByte() and SetClearAttr()

## Options:

<expN1> is the row to be erased. If not specified, the default is the current cursor line.

<expN2> is the column where the erasure is to begin. If not specified, the default is the current cursor position.

<expC3> is the color attribute used to clear. If not specified, the default is the attribute specified by SetClearAttr() and if this is not yet set, the current color set. <expC3> is comparable to the COLOR clause of @..SAY. Applicable for Terminal i/o mode and in GUI when SET GUICOLOR is ON, otherwise GUI mode uses the <expC5> color pair.

<expC4> is the character used to clear. If not specified, the default is the character specified by SetClearByte().

<expC5> is the color attribute same as <expC3> used in GUI mode when SET GUICOLOR is OFF. Comparable to GUICOLOR clause of @...SAY

## Returns:

<retC> is always a null string "".

## Description:

ClearEol() is similar to the standard command @ <expN1>,<expN2> or @ row(),col() which also clears the rest of specified line. In addition to, ClearEol() supports the use of different clear character (not only space) and/or using of different colors.

## Example:

```
/* clears rest of line from row(), col() by the character  
 * specified by SetClearByte() and color set by SetClearAttr()  
 */  
ClearEol()  
  
/* clears rest of line from 20, 50 to 20,MaxCol() using the  
 * character '#' and color red on blue in terminal i/o mode or  
 * red on gray in GUI mode.  
 */  
ClearEol(20,50, "R+/B", "#", "R+/W")
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only the first four arguments.

**Related:**

FS2:SetClearAttr(), FS2:SetClearByte(), FS2:GetClearAttr(), FS2:GetClearByte(), SetColor(), SET COLOR, @ row,col

# CLEARWIN ()

---

## Syntax:

```
retC = ClearWin ( [expN1], [expN2], [expN3],  
                  [expN4], [expNC5], [expNC6], [expC7] )
```

## Purpose:

Clears specified screen area, considering the clear byte and clear color attribute set by SetClearByte() and SetClearAttr()

## Options:

<expN1> is the top row coordinate of the screen or sub-window area to be cleared. If not specified, the current cursor row position is used.

<expN2> is the left column coordinate of the screen or sub-window area to be cleared. If not specified, the current cursor column position is used.

<expN3> is the bottom row coordinate of the screen or sub-window area to be cleared. If not specified, the MaxRow() value is used.

<expN4> is the right column coordinate of the screen or sub-window area to be cleared. If not specified, the MaxCol() value is used.

<expNC5> is the color attribute used to clear, specified as color string or a numeric equivalence (0..255) of. <expC5> is comparable to COLOR clause of @..SAY. If not specified, the default is the color pair specified by SetClearAttr(). Applicable for Terminal i/o mode and in GUI when SET GUICOLOR is ON, otherwise GUI mode uses the <expC6> color pair.

<expNC6> is the character used to clear, specified as string or a numeric equivalence (1..255) of. If not specified, the default is the character specified by SetClearByte().

<expC7> is the color attribute same as <expC5> used in GUI mode when SET GUICOLOR is OFF. Comparable to GUICOLOR clause of @...SAY

## Returns:

<retC> is always a null string "".

## Description:

ClearWin() is similar to the standard command @ <expN1>,<expN2> CLEAR TO <expN3>,<expN4> but ClearWin() supports the use of different clear character (not only space) and/or using of different colors.

## Example:

Clear rest of the screen (bottom right square) starting at current cursor position, using the "#" character and different colors for Terminal i/o and GUI mode:

```
Clearwin(,,, "R/B", "#", "R+")
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only the first six arguments.

**Related:**

FS2:SetClearAttr(), FS2:SetClearByte(), FS2:CleaEol(), @ row1,col1 CLEAR TO row2,col2

# CLEOL ()

---

**Syntax:**

```
retC = ClEol ( [expN1], [expN2] )
```

**Purpose:**

Clears from the cursor position to the end of line, without considering the clear byte and clear color attribute set by SetClearByte() and SetClearAttr()

**Options:**

<expN1> is the row to be erased. If not specified, the default is the current cursor line.

<expN2> is the column where the erasure is to begin. If not specified, the default is the current cursor column position.

**Returns:**

<retC> is always a null string "".

**Description:**

As opposite to ClearEol(), ClEol() does not consider the FS2 clear byte and clear color attribute. ClEol() is equivalent to the standard command @ <expN1>,<expN2> or @ row(),col().

**Example:**

```
ClEol(10, 15)      // both are
@ 10,15            // equivalent

ClEol()            // both are
@ row(), col()     // equivalent
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:ClearEol(), @ row,col

# CLWIN ()

---

**Syntax:**

**retC** = ClWin ( [**expN1**] , [**expN2**] , [**expN3**] , [**expN4**] )

**Purpose:**

Clears specified screen area, without considering the FS2 clear byte and clear color attribute.

**Options:**

<**expN1**> is the top row coordinate of the screen or sub-window area to be cleared. If not specified, the current cursor row position is used.

<**expN2**> is the left column coordinate of the screen or sub-window area to be cleared. If not specified, the current cursor column position is used.

<**expN3**> is the bottom row coordinate of the screen or sub-window area to be cleared. If not specified, the MaxRow() value is used.

<**expN4**> is the right column coordinate of the screen or sub-window area to be cleared. If not specified, the MaxCol() value is used.

**Returns:**

<**retC**> is always a null string "".

**Description:**

As opposite to ClearWin(), ClWin() does not consider the FS2 clear byte and clear color attribute. ClWin() is hence equivalent to the standard command @ <expN1>,<expN2> CLEAR TO <expN3>,<expN4>, but unlike the command, ClWin() does not need all four coordinates specified, it can determine them automatically.

**Example:**

```
ClWin() // both are
@ Row(),Col() CLEAR TO MaxRow(),MaxCol() // equivalent
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:ClearWin(), @ row1,col1 CLEAR TO row2,col2

# COLORTON ()

---

**Syntax:**

**retN = ColorToN ( expC1 )**

**Purpose:**

Converts a color pair string into numeric value.

**Arguments:**

<expC1> is a color pair coded as "nn/nn" or "cc/cc", see SET COLOR. If more pairs than the standard color is specified, only the first color pair is converted.

**Returns:**

<retN> is the numeric color value corresponding to the given color pair. This numeric value is coded as foreground-color + (background-color \* 16), e.g. "R+/GR" = "12/6" = 12 + 6\*16 = 108, or "W/N" = "7/0" = 7 + 0\*16 = 7. See SET COLOR for the numeric values. With invalid <expC1> value, 0 is returned.

**Description:**

ColorToN() converts a common color pair string (see SET COLOR) to numeric attribute used in some FS2 functions.

**Example:**

```
? ColorToN("07/00")      // 7
? ColorToN("7")           // 7
? ColorToN("7/N,R/B")     // 7
? ColorToN("W/N")         // 7

? ColorToN("12/06")       // 108
? ColorToN("12/6")        // 108
? ColorToN("R+/GR")       // 108
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:NtoColor(), SET COLOR, SetColor()

# DSETTYPE ( )

---

**Syntax:**

**retN = DsetType ( )**

**Purpose:**

Determines the size of the keyboard buffer = SET TYPEAHEAD

**Returns:**

<retN> is the current size of the keyboard buffer.

**Description:**

This function determines the currently set size of keyboard buffer as set to with SET TYPEAHEAD TO. The default size is 80.

**Example:**

```
? "current typeahead buffer size is", ltrim( DsetType() )  
SET TYPEAHEAD TO 1234  
? "new typeahead buffer size is", ltrim( DsetType() )
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

SET TYPEAHEAD, Set()



# ENHANCED ()

---

**Syntax:**

**retC = Enhanced ( )**

**Purpose:**

Selects the enhanced color value for SET COLOR TO output.

**Returns:**

<retC> is always "".

**Description:**

The color set with SET COLOR or SETCOLOR() can include different color pairs: the "standard", used in all screen output statements, the "enhanced" used in @..GET, READ, MENU, ACHOICE etc.

The Enhanced() function is equivalent to the command SETENHANCED and switches the current color pair so, that the corresponding color pair to become the "standard" one, Standard() function resets the original state.

**Example:**

```
SET COLOR TO "W+/B,R+/B,,,GR+/B"  
@ 1,2 say "hello with white"  
Enhanced()  
@ 1,col() say " continue with enhanced = red"  
Standard()  
@ 1,col() say " continue with standard"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:Standard(), FS2:Unselected(), SETENHANCED, SETSTANDARD

# GETCLEARA ()

## GETCLEARATTR ()

---

### Syntax 1:

```
retNC = GetClearAttr ( [expL1] )
```

### Syntax 2:

```
retNC = GetClearA ( [expL1] )
```

### Purpose:

Queries the current color attribute for clearing functions of FS2.

### Options:

<expL1> is an optional logical value. If set .T., the function returns the color attribute as common color string, e.g. "R+/B", otherwise as numeric equivalence.

### Returns:

<retN> is the currently set color attribute via SetClearAttr() returned as numeric equivalence (0..255) of the color pair or 0 if not set yet SetClearAttr().

<retC> is the currently set color attribute via SetClearAttr() returned as color pair ("foregr/backgr") when <expL1> is set .T. The default setting is "" if not set yet SetClearAttr().

### Description:

GetClearAttr() return the currently set status of a character used to clear part of a screen, specified by SetClearAttr().

### Example:

```
SetClearAttr("R+/N")

nClearColor := GetClearAttr()      // 12
cClearColor := GetClearAttr(.T.)   // "R+/N"
```

### Classification:

add-on library, programming

### Compatibility:

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only Syntax 2 without optional argument.

### Related:

FS2:SetClearAttr(), FS2:ClearEol(), SetColor(), SET COLOR

# GETCLEARB ()

## GETCLEARBYTE ()

---

### Syntax 1:

```
retNC = GetClearByte ( [expL1] )
```

### Syntax 2:

```
retNC = GetClearB ( [expL1] )
```

### Purpose:

Queries the current character used for the FS2 clearing function.

### Options:

<expL1> is an optional logical value. If set .T., the function returns the character as string, otherwise as numeric value.

### Returns:

<retN> is the currently set character via SetClearByte() returned as numeric equivalence (1..255) of the character. The default setting is 32 = space.

<retC> is the currently set character via SetClearByte() returned as string when <expL1> is set .T. The default setting is " ".

### Description:

GetClearByte() return the currently set character used to clear part of a screen, specified by SetClearByte().

### Example:

```
SetClearByte("#")  
  
nClearByte := GetClearByte()           // 35  
cClearByte := GetClearByte(.T.)        // "#"
```

### Classification:

add-on library, programming

### Compatibility:

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only the Syntax 2 without optional argument.

### Related:

FS2:SetClearByte(), FS2:ClearEol()

# FILESCREEN ()

---

**Syntax:**

**retN = FileScreen( expC1, [expN2] )**

**Purpose:**

Reads screen content from a file saved by ScreenFile().

**Options:**

<expC1> is the file name (optionally with path) where the screen content was saved by ScreenFile().

<expN2> specifies the file offset from which the screen is to be read. The default is 0, i.e. from the beginning of the file.

**Returns:**

<retN> is the number of bytes read. 0 or negative value indicates failure.

**Description:**

FileScreen() read the is screen content from thee specified file, previously saved by ScreenFile(), and restores the content on the screen, similar to standard RestScreen() function.

You may store several screen contents within the same file, see details in ScreenFile().

Notes: the by ScreenFile() created file is binary, not textual. The file content of Terminal i/o mode differs significantly from the content of GUI mode. The file created by FS2 is not compatible to file created by CA3.

**Example:**

```
FileScreen("myscreen.scr")           // restore screen
FileScreen("myscreen.png")           // restore GUI screen
FileScreen("myscreen.bmp", 98765")    // restore second screen
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. The file created by CA3 tools is not compatible to file created by FS2.

**Related:**

FS2:ScreenFile(), FS2:ScreenSize(), SaveScreen(), RestScreen()

# KEYSEND ()

---

**Syntax:**

**retL** = **KeySend** ( **expC1**, [**expL2**] )

**Purpose:**

Simulates keyboard buffer input.

**Options:**

<**expC1**> is a character or string to be put in keyboard buffer. The maximal stored buffer size can be determined by **DsetType()** or set by **SET TYPEAHEAD TO <size>**

<**expL2**> is an optional logical value specifying additive mode. If .T., the <expC1> string is added at the end of keyboard buffer, otherwise the keyboard buffer is overwritten by <expC1>.

**Returns:**

<**retL**> is .T. if all characters of <expC1> could be placed in the keyboard buffer, or .F. otherwise.

**Description:**

**KeySend()** works similarly to the **KEYBOARD** command.

**Example:**

```
keySend( chr(K_ESC) + "Hallo" + chr(K_PGUP), .T.)
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

**KEYBOARD** command, **FS2:DsetType()**, **Inkey()**, **inkey.fh**

# MAXCOL ()

# MAXROW ()

---

In NT2/CA3 tools, the MaxCol() and MaxRow() overlays the standard Clipper functions of the same name - but with different arguments.

To avoid misinterpretations, FS2 use WMaxCol() and WMaxRow() instead, which accepts also the optional logical parameter used by NT2/CA3.

The standard FlagShip functions MaxCol() and MaxRow() are equivalent to Clipper or NT2/CA3 functions of the same name when invoked without parameter.

FS2/FlagShip	<?>	NT2/CA3/Clipper	
MaxCol()	=	MaxCol()	size of selected window
WmaxCol()	=	MaxCol()	size of selected window
WmaxCol(.F.)	=	MaxCol(.F.)	size of selected window
WmaxCol(.T.)	=	MaxCol(.T.)	size of main window
WmaxCol(,0)	=	MaxCol(.T.)	size of main window
Max_Col()	=	MaxCol(.T.)	size of main window
MaxRow()	=	MaxRow()	size of selected window
WmaxRow()	=	MaxRow()	size of selected window
WmaxRow(.F.)	=	MaxRow(.F.)	size of selected window
WmaxRow(.T.)	=	MaxRow(.T.)	size of main window
WmaxRow(,0)	=	MaxRow(.T.)	size of main window
Max_Row()	=	MaxRow(.T.)	size of main window
MaxCol(.F.)	!=	MaxCol(.F.)	FS/FS2: size in cols
MaxCol(.T.)	!=	MaxCol(.T.)	FS/FS2: size in pixels
MaxRow(.F.)	!=	MaxRow(.F.)	FS/FS2: size in rows
MaxRow(.T.)	!=	MaxRow(.T.)	FS/FS2: size in pixels

**Related:**

FS2:WmaxCol(), FS2:WmaxRow(), MaxCol(), MaxRow()

# NUMCOL ()

---

**Syntax:**

**retN** = NumCol ( [expL1], [expN2] )

**Purpose:**

Return the number of columns in the current or specified sub- window or of the main screen.

**Options:**

<expL1> is an optional pixel specification. If .T., the returned coordinate is in pixel, if .F. the coordinate is col, otherwise the current SET PIXEL status is used.

<expN2> is the window ID# for which this function apply. If not specified, the currently selected sub-window is used.

**Returns:**

<retN> is the number of columns of the current or specified window.

**Description:**

NumCol() is very similar to WmaxCol() or to the standard MaxCol() function and differs to it only with the passed arguments, if such used, and with the returned value which is by one higher.

**Example:**

```
w1 := wopen(10,10, 20,50)
? wmaxCol(), MaxCol(), NumCol()           // 41 41 42
? wmaxCol(.T.), WMaxCol(.,0), NumCol(,0)  // 79 79 80
wselect(0)
? wmaxCol(), MaxCol(), NumCol()           // 79 79 80
wselect(w1)
wformat(1,1,1,1)
? wmaxCol(), MaxCol(), NumCol() // 39 39 40 ( 41 41 42 in GUI mode)
```

**Classification:**

add-on library, programming, windowing

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools does not supports the optional arguments.

**Related:**

FS2:WmaxCol(), FS2:Wopen(), FS2:Wselect(), FS2:Waclose(), MaxCol()

# NTOCOLOR ()

---

**Syntax:**

`retC = NtoColor ( expN1, [expL2] )`

**Purpose:**

Converts a numeric color value into a color pair string in the NN/NN or CC/CC form.

**Arguments:**

<expN1> is the numeric color value to be converted to color pair. This numeric value is coded as foreground-color + (background-color \* 16), e.g. "R+/GR" = "12/6" = 12 + 6\*16 = 108, or "W/N" = "7/0" = 7 + 0\*16 = 7. See SET COLOR for the numeric values.

**Options:**

<expL2> is an optional logical value, specifying conversion to color's character representation (if .T.) instead of the default numeric color representation.

**Returns:**

<retC> is the encoded color pair as "nn/nn" or "cc/cc" or "" if the numeric color value is invalid.

**Description:**

NtoColor() converts a color attribute returned from another FS2 functions in numeric form into the common alphanumeric data format, used in SET COLOR or SetColor().

**Example:**

```
? NtoColor(7)           // "07/00"
? NtoColor(7, .T.)      // "W/N"

? NtoColor(108)         // "12/06"
? NtoColor(108, .T.)    // "R+/GR"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:ColorToN(), SET COLOR, SetColor()



# SCREENFILE ( )

---

## **Syntax:**

```
retN = ScreenFile ( [expC1], [expL2], [expN3],  
                    [expL4], [expN5], [expN6] )
```

## **Purpose:**

Writes screen content to a file for a later restore by FileScreen().

## **Options:**

<expC1> is the file name (optionally with path) where the screen content should be saved.

<expL2> is an optional logical value specifying file overwriting or appending. .T. signals append = the new screen content will be added to the available file (if any). If .F. (the default), the available file content (if any) will be erased and replaced by the new screen content. If the given file is not available yet, this flag is meaningless and ignored.

<expN3> specifies the file offset from which the new screen is to be written. The default is end of the file. Considered only when appending, i.e. when <expL2> is .T.

<expL4> is optional logical value specifying trimming of the screen content in CA3tools. Ignored in FS2.

<expN5> is an optional numeric value specifying the file format and kind of compression for GUI mode. Supported values are: 1 = PNG, 2 = BMP, 3 = XBM, 4 = XPM, 5 = PNM, 6 = JPEG, 7 = GIF, or 0 (the default) using standard compression. Apply for GUI mode only, ignored otherwise.

<expN6> is an optional numeric value specifying the compression quality (if any). Supported values are 0 (lowest, max compression) to 100 (highest quality, no compression). If not specified, the default compression of the selected file format is used. Apply for GUI mode only, ignored otherwise.

## **Returns:**

<retN> is the number of bytes written. 0 or negative value indicates failure.

## **Description:**

ScreenFile() writes the content of main screen or of current sub- window (similarly to SaveScreen() function) to the specified file. It can be later read and restored by FileScreen(), similarly to the RestScreen() function.

Although not suggested, you may store several screen contents within the same file by specifying <expL2> = .T. and optionally <expN3>.

In GUI mode, the single-screen file corresponds to the given GUI file standard as given in <expN5>. If you choose the corresponding file extension, you will be able to display this file also by any other graphic software or with Web browser.

Notes: the by ScreenFile() created file is binary, not textual. The file content of Terminal i/o mode differs significantly from the content of GUI mode. The file created by FS2 is not compatible to file created by CA3.

**Example:**

```
ScreenFile("myscreen.png", , , , 1, 50)    // save screen
...
FileScreen("myscreen.png")                 // restore screen
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. The file created by CA3 tools is not compatible to file created by FS2. CA3tools supports only the first four arguments.

**Related:**

FS2: FileScreen(), FS2:ScreenSize(), SaveScreen(), RestScreen()

# SCREENSIZE ( )

---

## Syntax:

**retN = ScreenSize ( [expN1], [expN2], [expN3] )**

## Purpose:

Returns the size of screen image, i.e. the file size when saved by the same options via ScreenFile().

## Options:

**<expN1>** is either numeric value specifying the window ID# for which this function apply, in a range 0..Wnum(), or a logical .T. value specifying the main screen (same as numeric 0). If not specified, the currently selected screen or sub-window is used.

**<expN2>** is an optional numeric value specifying the file format and kind of compression for GUI mode. Supported values are: 1 = PNG, 2 = BMP, 3 = XBM, 4 = XPM, 5 = PNM, 6 = JPEG, 7 = GIF, or 0 (the default) using standard compression. Apply for GUI mode only, ignored otherwise. Correspond to **<expN5>** of ScreenFile().

**<expN3>** is an optional numeric value specifying the compression quality (if any). Supported values are 0 (lowest, max compression) to 100 (highest quality, no compression). If not specified, the default compression of the selected file format is used. Apply for GUI mode only, ignored otherwise. Correspond to **<expN6>** of ScreenFile().

## Returns:

**<retN>** is the size of the binary screen image and the file size when saved with the same options via ScreenFile().

## Description:

As opposite to ScreenSize([expL1]) of CA3Tools, which simply returns MaxRow() \* MaxCol() value, this FS2 function can be used when you wish to determine the size of a screen image to be saved by ScreenFile().

## Example:

```
nSize := ScreenSize(, 2)
if DiskSpace() < nSize
    ? "sorry, cannot save, " + ltrim(nSize) + " bytes not avail."
else
    ScreenFile("myscreen.bmp", , , , 2)    // save screen
endif
```

## Classification:

add-on library, programming

## Compatibility:

Available in FS2-Toolbox and in NT2/CA3 tools. The same named CA3tools supports only the first logical argument and have slightly different behavior.

**Related:** FS2: FileScreen(), FS2:ScreenFile()

# SETCLEARA ()

## SETCLEARATTR ()

---

### Syntax 1:

```
retC = SetClearAttr ( expNC1 )
```

### Syntax 2:

```
retC = SetClearA ( expNC1 )
```

### Purpose:

Sets the current color attribute for clearing functions of FS2.

### Argument:

<expN1> is a numeric equivalence (0..255) of the color pair used for clearing FS2 functions. The color pair is coded as foreground- color + (background-color \* 16), e.g. "R+/GR" = "12/6" = 12 + 6\*16 = 108, or "W/N" = "7/0" = 7 + 0\*16 = 7. See SET COLOR for the numeric values. You may clear the attribute by specifying 0, which then use the current SetColor().

<expC1> is alternatively a string containing the color pair used for clearing FS2 functions. See SET COLOR for further details. You may clear the attribute by specifying "", to use the current SetColor().

### Returns:

<retC> is always "".

### Description:

With SetClearAttr() you to set or modify the current, standard color attribute for screen-oriented FS2 functions. If SetClearAttr() was not set, the FS2 clear functions will use the current SetColor() colors.

You may retrieve the current setting by GetClearAttr().

### Example:

```
SetClearAttr("R+/N")
SetClearAttr(12)

nClearColor := GetClearAttr()      // 12
cClearColor := GetClearAttr(.T.)   // "R+/N"
```

### Classification:

add-on library, programming

### Compatibility:

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only Syntax 2 and has pre-defined "W/N" color pair.

### Related:

FS2:GetClearAttr(), FS2:ClearEol(), SetColor(), SET COLOR

# SETCLEARB () SETCLEARBYTE ()

---

**Syntax 1:**

```
retC = SetClearByte ( expNC1 )
```

**Syntax 2:**

```
retC = SetClearB ( expNC1 )
```

**Purpose:**

Sets the current character for clearing functions of FS2.

**Argument:**

<expN1> is a numeric equivalence (1..255) of the character used for clearing FS2 functions. The default is 32 = space.

<expC1> is alternatively a string containing the character used for clearing FS2 functions. If the string length is > 1, only the first character is used. The default is space " ".

**Returns:**

<retC> is always "".

**Description:**

With SetClearByte() you to set or modify the current character used for screen-oriented FS2 functions. If SetClearByte() was not set, the FS2 clear functions will use space.

You may retrieve the current setting by GetClearByte().

**Example:**

```
SetClearByte("#")  
SetClearByte(35)  
  
nClearByte := GetClearByte()           // 35  
cClearByte := GetClearByte(.T.)        // "#"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. CA3tools supports only Syntax 2 and has pre-defined chr(255).

**Related:**

FS2:GetClearByte(), FS2:ClearEol()

# STANDARD ( )

---

**Syntax:**

`retC = Standard ( )`

**Purpose:**

Selects the standard color pair for SET COLOR TO output.

**Returns:**

<retC> is always "".

**Description:**

The color set with SET COLOR or SETCOLOR() can include different color pairs: the "standard", used in all screen output statements, the "enhanced" used in @..GET, READ, MENU, ACHOICE etc.

The Standard() function is equivalent to the command SETSTANDARD and switches the current color pair so, that the corresponding color pair to become the "standard" one, e.g. to reset the color pair from Enhanced() to the original state.

**Example:**

```
SET COLOR TO "W+/B,R+/B,,,GR+/B"
@ 1,2 say "hello with white"
Enhanced()
@ 1,col() say " continue with enhanced = red"
Standard()
@ 1,col() say " continue with standard"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:Enhanced(), FS2:Unselected(), SETENHACED, SETSTANDARD

# TRAPANYKEY ()

---

**Syntax:**

```
retB = TrapAnyKey ( [expCB1] )
```

**Purpose:**

Calls a procedure or codeblock with any keyboard input.

**Options:**

<expCB1> is either a name of user defined function or a code block that is called when any input key is touched. If <expCB1> is not specified or is NIL, the trapping is disabled.

**Returns:**

<retB> is the code block previously set by TrapAnyKey() or by ON ANY KEY, or is NIL if none was set.

**Description:**

TrapAnyKey() is very similar to the standard FlagShip command ON ANY KEY TO <expC1> or to the OnKey(0, <expB1>) function and evaluates the assigned code block or user defined function on any key.

It sets or retrieves the automatic action associated with any key during a wait status. A wait status is any mode that extracts keys from the keyboard except for INKEY(), but including ACHOICE(), INKEYTRAP(), DBEDIT(), MEMOEDIT(), ACCEPT, INPUT, READ and WAIT.

When an assigned key is pressed during a wait state, the EVAL() function evaluates the associated action block, passing the parameters PROCNAME(), PROCLINE(), READVAR() and the numeric value of the pressed key, in that order.

**Example:**

```
TrapAnyKey( "MyUdf" ) // both are
TrapAnyKey( {|p1,p2,p3,p4| MyUdf(p1,p2,p3,p4) } ) // equivalent
wait
TrapAnyKey()

// protocol all key pressed except inkey()
FUNCTION MyUdf(cProc, nLine, cReadVar, nKey)
    set printer on
    set console off
    ? time(), "key:", ltrim(nKey), " = ", inkey2str(nKey)
    set console on
    set printer off
    return NIL
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox and in NT2/CA3 tools. CA3tools supports only the <expC1> argument, not <ecpB1>, passes only one argument (the key value) to the UDF, and returns a string instead of code block.

**Related:**

ON ANY KEY, OnKey(), SET KEY



# UNSELECTED ()

---

**Syntax:**

`retC = Unselected ( )`

**Purpose:**

Selects the unselected color pair for SET COLOR TO output.

**Returns:**

<retC> is always "".

**Description:**

The color set with SET COLOR or SETCOLOR() can include different color pairs: the "standard", used in all screen output statements, the "enhanced" used in @..GET, READ, MENU, ACHOICE etc. or "unselected", used in the READ command.

The Unselected() function is equivalent to the command SETUNSELECTED and switches the current color pair so, that the corresponding color pair to become the "standard" one, the Standard() function resets the original state.

**Example:**

```
SET COLOR TO "W+/B,R+/B,,,GR+/B"  
@ 1,2 say "hello with white"  
Unselected()  
@ 1,col() say " continue with unselected = yellow"  
Standard()  
@ 1,col() say " continue with standard"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

FS2:Standard(), FS2:Enhanced(), SETSTANDARD, SETUNSELECTED

# Serial Communication

---

## Introduction

With the FS2 Toolbox Serial Communication functions, you can control and communicate not only via serial port, but also via any other peripheral interface.

The functions in this chapter offer possibilities for data transmission and allow you to influence control signals. These functions do not support any particular protocol nor any specific instrument.

To be able to easily adapt the communication protocol to your needs, the significant part is available in **source code** in the `fs2_com.prg` file located in `<FlagShip_dir>/system` directory.

## Index of FS2 Serial Communication Functions

* COM_BREAK()	Creates a break on a transmission line
COM_CLOSE()	Clears the receiving buffer and closes the com port
COM_COUNT()	Counts the number of characters in the input buffer
COM_CRC()	Computes a Cyclic Redundancy Check (CRC) for the string
* COM_CTS()	Queries the Clear To Send (CTS) status
* COM_DCD()	Queries the Data Carrier Detect (DCD) status
* COM_DOSCON()	Provides screen output through DOS ANSI.SYS terminal emulation
* COM_DSR()	Queries the Data Set Ready (DSR) status
* COM_DTR()	Queries/sets the Data Terminal Ready (DTR) status
COM_ERROR()	Reports the last r/w error, if any
* COM_ERRCHR()	Defines replacement for a character not received correctly
* COM_EVENT()	Designates which event at the port triggered a key trap
COM_FLUSH()	Clears the receiving buffer
COM_GETIO()	Determines the base address of a port
COM_GETIRQ()	Determines the interrupt request for a port
COM_HARD()	Turns the hardware handshake (automatic CTS) on/off
COM_INIT()	Initializes the port parameters
COM_INIT2()	Initializes a subset of port parameters
COM_ISOPEN()	Checks if the port is already open
* COM_KEY()	Monitors the port using key traps
COM_LSR()	Reads the Line Status Register (LSR)
* COM_MCR()	Reads or sets the Modem Control Register (MCR)
* COM_MSR()	Reads the Modem Status Register (MSR)
COM_NUM()	Gives the number of the highest available serial interface port
COM_OPEN()	Opens the port and initializes the buffer
COM_READ()	Reads characters from the receiving buffer
* COM_REMOTE()	Determines the clear character for the receiving buffer
* COM_RING()	Queries the ring line

* COM_RTS()	Queries or sets the Request To Send (RTS)
COM_SCOUNT()	Counts number of characters in the background sending buffer
COM_SEND()	Transmits data directly or in the background
* COM_SETIO()	Changes the base address for a port
* COM_SETIRQ()	Changes the interrupt request for a port
COM_SFLUSH()	Deletes the sending background buffer
* COM_SKEY()	Monitors port using key traps during background transmission
COM_SMODE()	Determines the current status of a background transmission
COM_SOFT()	Queries or sets the software handshake (automatic XON/XOFF)
COM_SOFT_R()	Tests to see if an XOFF character has been received
* COM_SOFT_S()	Tests to see if buffer has automatically sent an XOFF character
XMOBLOCK()	Generates a block for XMODEM transmission
XMOCHECK()	Tests a received XMODEM block
ZEROINSERT()	Inserts a 0-bit after every fifth 1-bit
ZEROREMOVE()	Removes 0-bits in a file block

Functions marked by (\*) are available for compatibility purposes only in the source file fs2\_com.prg. When FS\_SET("devel".T.) is set, you will receive developer's warning, otherwise default return value only. These functions are not applicable in Unix nor for 32bit Windows nor are cross-portable and should therefore not be used for a new development.

# Disk, File and Database Functions

---

## Introduction

These functions gives you additional information and access to files and directories. FS2 Tools provides all the corresponding functions from CA3 Tools. But several of them, marked with a star, are too DOS specific and are therefore emulated only in Unix and/or multi-tasking & multi-user environment. The source code of these emulated functions is available in the source file fs2\_udfs.prg and can hence be easy modified.

## Index of FS2 Disk and File Functions

CSETSAFETY()	Queries/sets the safety mode switch (see FS2 Setting Funct)
DELETEFILE()	Deletes an error-tolerant file
DIRCHANGE()	Changes the current directory
DIRMAKE()	Creates a directory
DIRNAME()	Determines the name of the current directory
DIRREMOVE()	Removes a directory
* DISKCHANGE()	Changes the current disk drive
* DISKCHECK()	Creates a checksum for a disk
* DISKFORMAT()	Formats disks, controlled through a UDF
DISKFREE()	Determines the space available on a floppy or hard disk
* DISKNAME()	Determines the drive designator for the current drive
* DISKREADY()	Tests to see if a disk drive is ready
* DISKREADYW()	Queries whether you can write to a drive
* DISKSPEED()	Determines a comparison value for the drive speed
* DISKSTAT()	Determines the status of a drive
DISKTOTAL()	Determines the total capacity of a floppy or hard disk
* DISKTYPE()	Determines the type of data carrier
* DRIVETYPE()	Determines the drive type
FILEAPPEND()	Appends data to a file
FILEATTR()	Determines a file's attributes
* FILECCLOSE()	Closes a file after backup mode
* FILECCONT()	Copies sections of a file in backup mode
* FILECDATI()	Determines which date the target file contains with FILECOPY()
FILECHECK()	Calculates/computes/determines a checksum for a file
* FILECOPEN()	Tests to see if the file is still open in the backup mode
FILECOPY()	Copies files normally or in backup mode
FILEDATE()	Determines the file date
FILEDELETE()	Deletes file(s) by name and attribute
FILEGROUP()	Determines the group of a file
FILEMOVE()	Moves files to another directory
FILEOWNER()	Determines the owner of a file
FILERIGHTS()	Determine or set file access rights (permission)
FILESEEK()	Searches for files by name and attribute

FILESIZE()	Determines the size of a file
FILESTR()	Reads a portion of a file into a string
FILESYMLNK()	Determines the parent file name of a symbolic link
FILETIME()	Determines a file's time
FILEVALID()	Tests whether a string has a valid file name
* FLOPPYTYPE()	Determines the exact type of floppy drive
* GETSHARE()	Determines the file open (share) mode
* NUMDISKF()	Determines the number of installed disk drives
* NUMDISKH()	Determines the number of hard disks
* NUMDISKL()	Determines the number of available logical drives
RENAMEFILE()	Fault tolerant renaming of a file
RESTFSEEK()	Restores the FILESEEK environment
SAVEFSEEK()	Saves the current FILESEEK environment
SETFATTR()	Sets a file's attributes
SETFCREATE()	Sets default file attribute for creating with FS2 functions
SETFDATI()	Sets the date and time of a file
* SETSHARE()	Sets default opening mode for FS2 file functions
STRFILE()	Writes a string to a file
TEMPFILE()	Creates a file for temporary use
TRUENAME()	Standardizes the path designation
* VOLSERIAL()	Determines the DOS disk serial number
* VOLUME()	Establishes a volume label for a floppy or hard disk

## Index of Database Functions

DBFDSKSIZE()	Determines the size of a selected (.dbf) file on a disk drive
DBFSIZE()	Determines the size of a selected (.dbf) file in memory
+ FIELDDECI()	Determines the number of decimal places in a field
FIELDNUM()	Determines field number for a specific field in a database
FIELD SIZE()	Determines the size of a field
+ FIELDTYPE()	Determines the data type for a field
ISDBT()	Determines if a memo file (.dbt) is present

Functions marked by (+) are available in the standard FlagShip library.

Functions marked by (\*) are available for compatibility purposes only in the source file fs2\_udfs.prg. These functions are not applicable in Unix nor are cross-portable and should therefore not be used for a new development. When FS\_SET("devel",.T.) is set, you will receive developer's warning, otherwise default return value only.

# DELETEFILE ()

---

## Syntax:

**retN = DeleteFile ( expC1 )**

## Purpose:

Deletes an error-tolerant file.

## Arguments:

<expC1> is the file name, optionally with path, to delete. Wildcards are not supported. The file name is used as is, i.e. FS\_SET("path...") and FS\_SET("lower") is not considered, but you may use TRUEPATH() to convert <expC1> to absolute path.

## Returns:

<retN> is a success or error code:

0	FA_NO_ERROR	No error occurs
-1	FA_FILE_ERROR	Invalid parameter
-2	FA_FILE_NOT_FOUND	File not found
-3	FA_PATH_NOT_FOUND	Path not found
-5	FA_ACCESS_DENIED	Access denied (e.g. the file or directory is read-only)

where the FA\_\* constants are defined in fileio.fh

## Description:

DeleteFile() unrecoverable removes the file <expC1> from hard disk. It is nearly equivalent to FS2:FileDelete() and differs only in the return value but DeleteFile() requires the <expC1> argument and does not support wildcards, while FileDelete() can remove also files group specified by wildcard.

DeleteFile() is similar to the standard FlagShip function Ferase() or to the DELETE FILE command.

## Example:

```
nRet := DeleteFile("../test/myfile.tmp")
if nRet != 0
    ? "could not delete file ../test/myfile.tmp"
endif
```

## Example:

```
if File("temp.dbf")                // database file available?
    USE temp EXCL NEW                // yes, try to open exclusive:
    if used()                        // success means the database
        USE                          // is not in use by others
        DeleteFile("temp.dbf")      // or: Ferase("temp.dbf")
    else
        ? "Could not delete temp.dbf, database is in use"
    endif
else
    ? "File temp.dbf not available"
endif
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

Ferase(), DELETE FILE, File(), FS2:DirRemove()

# DIRCHANGE ()

---

## Syntax:

```
retN = DirChange ( expC1, [expL2] )
```

## Purpose:

Changes the current directory.

## Arguments:

<expC1> is the path of the directory that is changed. In MS-Windows, you may specify drive: and path

## Options:

<expL2> If this parameter is .F., the file <expC1> is used "as is". If <expL2> is not given or is .T. (the default), the path name in <expC1> is translated to lower/uppercase according to FS\_SET("pathlower"|"pathupper") or #include "fspreset.fh" setting, and the DOS drive mapping via ?\_FSDRIVE is considered accordingly.

## Returns:

<retN> is a success or error code:

0	FA_NO_ERROR	No error occurs
-1	FA_FILE_ERROR	Invalid parameter
-3	FA_PATH_NOT_FOUND	Path not found
-5	FA_ACCESS_DENIED	Access denied (e.g. the file or directory is read-only)

where the FA\_\* constants are defined in fileio.fh

## Description:

DirChange() changes the current directory and returns 0 on success. It is comparable to SET DIRECTORY command, but you may check the success here.

With set x\_FSDRIVE environment variable (see LNG.9.5 and FSC.3.3), you also may use DOS drives for <expC1>, mapped then to desired Unix/Linux path, except when this feature is disabled by <expL2>.

## Example:

```
if DirChange("data") == 0
    ? "Current directory is now", Dirname()
else
    ? "directory", DirName() + "/data is not available", ;
    "or has insufficient permission"
endif
```

## Example:

```
// assuming you have set "C_FSDRIVE=/tmp ; export C_FSDRIVE", see
// LNG.9.5 and FSC.3.3 and have available the /tmp/data directory

#include "fspreset.fh"
cMap := getenv("C_FSDRIVE")
? "Environment variable C_FSDRIVE is '" + cMap + "'"
```



```

? "Current directory is", Dirname()

if DirChange("C:\data") == 0
    ? "Current directory changed to", Dirname()
else
    ? "Sorry, directory", cMap + "/data is not available,"
    ? "or environment variable C_FSDRIVE is not set properly"
endif
wait

```

**Example:** on MS-Windows

```

if DirChange("C:") == 0
    ? "current dir changed to", curdir()
endif
if DirChange("D:\tmp") == 0
    ? "current dir changed to", curdir()
endif
// if set on command line: SET X_FSDRIVE=D:
if !empty(getenv("X_FSDRIVE")) .and. DirChange("x:") == 0
    ? "current dir changed for mapped drive to", curdir()
endif

```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools, which supports the first parameter only.

**Related:**

CurDir(), FS2:DirMake(), FS2:DirName(), SET DIRECTORY

# DIRMAKE ()

---

**Syntax:**

```
retN = DirMake ( expC1, [expL2] )
```

**Purpose:**

Creates a directory.

**Arguments:**

<expC1> is the name of the directory to create

**Options:**

<expL2> If this parameter is .F., the path <expC1> is used "as is". If <expL2> is not given or is .T. (the default), the path name in <expC1> is translated to lower/uppercase according to FS\_SET("pathlower"|"pathupper") or #include "fspreset.fh" setting, and the DOS drive mapping via ?\_FSDRIVE is considered accordingly.

**Returns:**

<retN> is a success or error code:

0	FA_NO_ERROR	No error occurred, directory exist or was successfully created
-1	FA_FILE_ERROR	wrong argument
-5	FA_ACCESS_DENIED	Access denied (e.g. the parent directory is read-only)

where the FA\_\* constants are defined in fileio.fh

**Description:**

DirMake() allows you to create new directories from within the application. As opposite to the command "mkdir" which allows to create directories in available parents only, DirMake() will create the parents too when required and considers the automatic translation via FS\_SET("pathlower"|"pathupper") as well as mapping of DOS drives via ?\_FSDRIVE environment vars, see LNG.9.5 and FSC.3.3

**Example:**

```
? DirName() // /home/john/applic
if DirChange("data") != 0 // subdirectory not found,
    DirMake("data") // create it
    if DirChange("data") == 0
        ? "Directory " + DirName() + " created successfully"
    else
        alert("cannot access " + CurDir() + "/data;" + ;
            "check directory permission")
        quit
    endif
endif
? DirName() // /home/john/applic/data
use mydbf shared // /home/john/applic/data/mydbf.dbf
```

**Example:**

```
FS_SET("pathlower", .T.)
cNewdir := "data/TeSt/test2"
ok = DirMake(cNewdir)
if ok == 0
    ? "directory", lower(cNewdir), "created successfully"
else
    ? "sorry, insufficient permission to create", lower(cNewdir)
endif
ok = DirChange(cNewDir)
if ok != 0
    ? "sorry, insufficient permission to access", lower(cNewdir)
endif
? "current directory is", DirName()
```

**Example:**

```
// assuming C_FSDRIVE maps to /tmp
? "C_FSDRIVE is set to '" + getenv("C_FSDRIVE") + "'"

ok1 := DirMake("C:\data\\hugo")
ok2 := DirChange("C:\data\\hugo")
if ok2 != 0
    ? "sorry, insufficient permission/mapping to access", ;
      "C:\data\\hugo = ", TruePath("C:\data\\hugo")
endif
? "current directory is", DirName()    // here: /tmp/data/hugo
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools, which supports the first parameter only.

**Related:**

CurDir(), FS2:DirMake(), FS2:DirName()

# DIRNAME ()

---

## Syntax:

```
retC = DirName ( [expC1] )
```

## Purpose:

Determines the name of the current directory

## Options:

<expC1> is the DOS drive with or w/o colon for which the current directory mapping is determined. On Linux, considered only when the corresponding ?\_FSDRIVE (see LNG.9.5 and FSC.3.3) is set. With MS- Windows, you may specify drive: with path.

## Returns:

<retC> is the current directory name or "" on error, e.g. if the drive in <expC1> is not mapped via x\_FSDRIVE, or this mapped directory is not available or not accessible. In MS-Windows, DirName(...) is similar to DIR ... command, i.e. "drive:path" of the specified drive is returned, if available.

## Description:

DirName() determines the current directory (or the DOS directory mapped by x\_FSDRIVE). It is similar to standard function CURDIR() but allows specifying of DOS drive.

With set x\_FSDRIVE environment variable (see LNG.9.5 and FSC.3.3), you also may use DOS drives for <expC1>, mapped then to desired Unix/Linux path or Windows drive.

## Example:

```
? "Current directory is", DirName()
if DirChange("../data") == 0
  ? "Current directory changed to", Dirname()
endif
```

## Example:

```
// assuming you have set "C_FSDRIVE=/tmp ; export C_FSDRIVE", see
// LNG.9.5 and FSC.3.3 and have available the /tmp/data directory

cMap := getenv("C_FSDRIVE")
? "Environment variable C_FSDRIVE is '" + cMap + "'"
? "Current directory is", DirName()
? "Mapped directory to 'C:' drive is", DirName("C") // "/tmp"
? "Mapped directory to 'D:' drive is", DirName("D:") // "" = n/a
```

## Example: on MS-Windows

```
? CurDir() // "C:\my\own\path"
DirChange("D:\tmp")
? DirName() // "D:\tmp"
? DirName("C") // "C:\my\own\path"
? DirName("C:") // "C:\my\own\path"
? DirName("C:\") // "C:\"
? DirName("C:/my/own") // "C:\my\own"
? DirName("C:\unknown") // "" (empty, n/a)
```

```
? DirName("E:")           // "E:\" (or empty if n/a)

// if set on command line: SET X_FSDRIVE=D:
if empty(getenv("X_FSDRIVE"))
    ? "mapped drive X: is not set"
endif
? DirName("X:")           // "D:\tmp" (or empty if n/a)
? DirName("x")           // "D:\tmp" (or empty if n/a)
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:**

CurDir(), FS2:DirMake(), FS2:DirChange(), SET DIRECTORY

# DIRREMOVE ()

---

## Syntax:

**retN = DirRemove ( expC1 )**

## Purpose:

Removes a directory

## Arguments:

<expC1> is the directory name (optionally with path) to remove. The name is used as is, i.e. FS\_SET("path...") is not considered, but you may use TRUEPATH() to convert <expC1> to absolute path.

## Returns:

<retN> is a success or error code:

0	FA_NO_ERROR	No error occurs
-1	FA_FILE_ERROR	Invalid parameter
-3	FA_PATH_NOT_FOUND	Path not found or directory not empty
-5	FA_ACCESS_DENIED	Access denied (e.g. the directory has no write access or is not owned by curr user)
16	FA_CURRENT_DIR	<expC1> is current directory (cannot be removed)

where the FA\_\* constants are defined in fileio.fh

## Description:

DirRemove() removes the given directory. On success, 0 is returned. This function handles similarly to the system command "rmdir" which allows you to remove empty directory only. If you wish to remove non-empty directory unconditionally, you may alternatively issue RUN("rm -rf /dir/name") but use this VERY carefully and be aware what you are doing!

## Example:

```
? "Current directory is", DirName() // /home/user/john/src
xName := TruePath("../data") // /home/user/john/data
ok := DirRemove("../data") // or: DirRemove(xName)
if ok == 0
    ? "Directory " + xName + " removed"
else
    ? "Cannot remove directory ../data, error code:", ltrim(ok)
endif
```

## Classification:

add-on library, programming

## Compatibility:

Available in FS2-Toolbox, compatible to NT2/CA3 tools.

## Related:

CurDir(), FS2:DirName(), FS2:DirMake(), FS2:DirChange(), TruePath()

# DISKFREE ()

---

**Syntax:**

**retN = DiskFree ( [expC1], [expL2] )**

**Purpose:**

Determines the space yet available on a hard disk, floppy, file system or network drive

**Options:**

<expC1> is path containing the name of the file system you want to determine the free space for. DOS drives are supported as well, when ?\_FSDRIVE environment variable is set, see LNG.9.5 and FSC.3.3 If the argument is not given or is not of the character type, the default is the current file system.

<expL2> If this parameter is .F., the path <expC1> is used "as is". If <expL2> is not given or is .T. (the default), the path name in <expC1> is translated to lower/uppercase according to FS\_SET("pathlower"|"pathupper") or #include "fspreset.fh" setting, and the DOS drive mapping via ?\_FSDRIVE is considered accordingly.

**Returns:**

<retN> is a numeric value representing the number of free bytes remaining on the specified file system. On error, 0 is returned.

**Description:**

DiskFree() is similar to the standard function DISKSPACE(). It is used to determine the amount of free space on selected or default hard disk (or network) drive.

**Example:**

```
? "Current directory is", DirName()  
? "and there are", ltrim(DiskFree()), ;  
  "bytes free on this drive/file system"  
  
? "The root file system /* contains total", ;  
  ltrim(str((DiskTotal("/") / (1024 * 1024 * 1024)), 10, 3)), ;  
  "GB, still available are", ;  
  ltrim(str((DiskFree("/") / (1024 * 1024 * 1024)), 10, 3)), "GB"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools which supports only the first parameter.

**Related:**

CurDir(), FS2:DiskTotal(), FS2:DirName(), TruePath()

# DISKTOTAL ()

---

**Syntax:**

`retN = DiskTotal ( [expC1], [expL2] )`

**Purpose:**

Determines the total space of a hard disk, floppy, file system or network drive

**Options:**

**<expC1>** is path containing the name of the file system you want to determine the free space for. DOS drives are supported as well, when `?_FSDRIVE` environment variable is set, see LNG.9.5 and FSC.3.3 If the argument is not given or is not of the character type, the default is the current file system.

**<expL2>** If this parameter is `.F.`, the path `<expC1>` is used "as is". If `<expL2>` is not given or is `.T.` (the default), the path name in `<expC1>` is translated to lower/uppercase according to `FS_SET("pathlower"|"pathupper")` or `#include "fspreset.fh"` setting, and the DOS drive mapping via `?_FSDRIVE` is considered accordingly.

**Returns:**

**<retN>** is a numeric value representing the number of total bytes available on (or allocated to) the specified file system. On error, 0 is returned.

**Description:**

`DiskTotal()` is used to determine the total size of a harddisk or floppy or network drive.

**Example:**

```
? "Current directory is", DirName()  
? "and there are", ltrim(DiskFree()), "bytes free on this"  
? "drive/file system from total", ltrim(DiskTotal()), "bytes"
```

**Classification:**

add-on library, programming

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools which supports only the first parameter.

**Related:**

`CurDir()`, `FS2:DiskFree()`, `FS2:DirName()`, `TruePath()`



# FILEAPPEND ()

---

**Syntax:**

**retN = FileAppend ( expC1, expC2 )**

**Purpose:**

Appends data to a file

**Arguments:**

<expC1> is the source file name (optionally prefaced by path) that is appended to <expC2>. The file name is used as is, i.e. FS\_SET("path...") is not considered, but you may use TRUEPATH() to convert <expC1> to absolute path.

<expC2> is the designation file name (optionally prefaced by path) to which <expC1> is appended. The file name is used as is, i.e. FS\_SET("path...") is not considered, but you may use TRUEPATH() to convert <expC2> to absolute path.

**Returns:**

<retN> is the number of characters/bytes appended to <expC2>

**Description:**

FileAppend() reads the file <expC1> and appends all data to file <expC2>. It is used e.g. to concatenate splitted files. The file attribute of <expC2> is set to the SetFcreate() rights (default is FA\_RW\_ALL = read/write for all) if a new file is created.

**Classification:**

add-on library, programming

**Example:**

```
// concatenate file1.part ... file5.part into one file.data
cDest := "file.data"
FileDelete(cDest)
for iPos := 1 to 5
    cSrc := "file" + ltrim(iPos) + ".part"
    if !file(cSrc)
        alert("File " + cSrc + " not available")
        quit
    endif
    if FileAppend(cSrc, cDest) < 0
        alert("could not write file " + cDest)
    endif
next
FileRights(cDest, "rw-rw-r--", .F.)
? "File", cDest, "created, total", ltrim(FileSize(cDest)), "bytes"
```

**Compatibility:** Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:** FS2:SETFCREATE(), FS2:FILECOPY(), FS2:FILEMOVE(), FS2:FILEDELETE(), FS2:FILEATTR(), FS2:FILESEEK(), FS2:FILESIZE(), FS2:FLERIGHTS(), FS2:FILEOWNER(), FS2:FILEGROUP(), FS2:FILESYMLNK(), FILE()

# FILEATTR ()

---

**Syntax:**

```
retN = FileAttr ( expC1 )
```

**Purpose:**

Determines a file's attributes

**Arguments:**

<expC1> is the file name (optionally prefaced by path) which access rights has to be determined. The file name is used as is, i.e. FS\_SET("path...") is not considered, but you may use TRUEPATH() to convert <expC1> to absolute path. If <expC1> is not specified, the attribute for the most-recent file located with FILESEEK(), if any, is returned.

**Returns:**

<retN> is the status of <expC1> file or 0 on error. In Unix, the return value represents a decimal equivalent of file status (see "man 2 fstat"), including the file type (32768 or octal 100000 is regular file) and it access rights, e.g. 33204 = octal 100664 for "rw-rw-r--" of a regular file. The single flags can be determined by BinAnd() or IsBit(), see example. The attribute flags are:

Value-deci	octal	IsBit()	Constant	Assigned attribute
49152	0140000	16+15		socket
40960	0120000	16+14		symbolic link
32768	0100000	16		regular file
24576	0060000	15+14		block device
16384	0040000	15	FA_DIRECTORY	directory
8192	0020000	14		character device
4096	0010000	13		fifo
2048	0004000	12		set UID bit (use with care!)
1024	0002000	11		set GID bit (use with care!)
512	0001000	10		sticky bit (use with care!)
493	0000755	9+8+..	FA_EXEC_ALL	execute for all = rwxr-xr-x
438	0000666	9+8+..	FA_RW_ALL	read/write for all = rw-rw-rw-
420	0000644	9+8+..	FA_NORMAL	read/write owner = rw-r--r--
256	0000400	9	FA_READONLY	owner has read permission
128	0000200	8	FA_WRITEONLY	owner has write permission
64	0000100	7	FA_EXECUTE	owner has execute permission
32	0000040	6		group has read permission
16	0000020	5		group has write permission
8	0000010	4		group has execute permission
4	0000004	3		others have read permission
2	0000002	2		others have write permission
1	0000001	1		others have execute permission

In MS-DOS, following attributes are returned:

Value	octal	IsBit()	Constant	Assigned attribute
0	0000000		FA_NORMAL	Normal (regular file)
0	0000000		FA_RW_ALL	Normal
0	0000000		FA_EXEC_ALL	Normal
1	0000001	1	FA_READONLY	READ ONLY (Read-only)
2	0000002	2	FA_HIDDEN	HIDDEN (Hidden files)
4	0000004	3	FA_SYSTEM	SYSTEM (System files)
8	0000010	4	FA_VOLUME	VOLUME (Name of a floppy/hard disk)
16	0000020	5	FA_DIRECTORY	DIR (Directory)
32	0000040	6	FA_ARCHIVE	ARCHIVE (Changes since last backup)

where the corresponding file attributes are added to each other and can also be separated by BinAnd() or IsBit(). The symbolic constants are defined in fileio.fh

**Description:**

FileAttr() determines the current status of the given file. It is similar to FileRights() which determines access rights of regular files. If the <expC1> is a symbolic link, <retN> reports status of the link, not of the parent file.

**Classification:**

add-on library, programming

**Example:**

```
cFile := "test.txt"
if !file(cFile)
    run ("touch " + cFile)                // create
    ? "created: ",cFile,"=", FileRights(cFile) // rw-r--r--
endif

nAttr := FileAttr(cFile)
? "The file " + cFile + " is a "
if BinAnd(nAttr, 32768) == 32768
    ?? "regular file"
elseif BinAnd(nAttr, 40960) == 40960
    ?? "symbolic link"
elseif BinAnd(nAttr, 16384) == 16384
    ?? "directory"
else
    ?? "special file"
endif
?? " and the owner has "
if IsBit(nAttr, 9) .and. IsBit(nAttr, 8)
    ?? "read/write"
elseif IsBit(nAttr, 9)
    ?? "read only"
elseif IsBit(nAttr, 8)
    ?? "write only"
else
    ?? "NOT read nor write"
endif
?? " access rights."
```

**Example:**

see also FS2:SaveFseek() for additional examples

**Compatibility:**

Available in FS2-Toolbox, compatible NT2/CA3 tools (return values differs between DOS and Unix).

**Related:**

FS2:FILERIGHTS(), FS2:FILESEEK(), FS2:FILESIZE(), FS2:FILEDATE(),  
FS2:FILETIME(), FS2:FILEOWNER(), FS2:FILEGROUP(), FS2:FILESYMLNK(),  
DIRECTORY()

# FILECHECK ()

---

**Syntax:**

**retN = FileCheck ( expC1, [expL2] )**

**Purpose:**

Calculates a checksum for a file.

**Arguments:**

<expC1> is the file name (optionally prefaced by path) which checksum has to be determined. The file name is used as is, i.e. FS\_SET("path...") is not considered, but you may use TRUEPATH() to convert <expC1> to absolute path.

**Options:**

<expL2> If this parameter is .F., the file name <expC1> is used "as is". If <expL2> is not given or is .T. (the default), the file name (and path) in <expC1> is translated to lower/uppercase according to FS\_SET("lower"|"pathlower"|...) or #include "fspreset.fh" setting, and the DOS drive mapping via ?\_FSDRIVE is considered accordingly.

**Returns:**

<retN> is a checksum value for <expC1> or -1 on error.

**Description:**

FileCheck() calculates checksum value of the given file using CRC32 algorithm. To be able to open the file, at least read access rights are required. Embedded zero bytes within the file are supported (i.e. you also may check binary files and executables).

**Example:**

```
#include "fspreset.fh"           // includes FS_SET("lower", .T.)
PUBLIC nSavedCheck := -1
if file("laststatus.mem")
    RESTORE FROM lastStatus      // retrieve last file status
endif
// ...
if nSavedCheck != -1 .and. FileCheck("data.dbf") != nSavedCheck
    alert("warning: file data.dbf has been irregularly changed!")
endif
// process ...
nSavedCheck := FileCheck("data.dbf")
SAVE ALL LIKE nSaved* TO lastStatus  // save current status
quit
```

**Classification:** add-on library, programming

**Compatibility:** Available in FS2-Toolbox, compatible to NT2/CA3 tools (except the return value where the CA3 value is compatible to itself only). The second parameter is supported by FS2 only.

**Related:** CRC32(), FS2:CRC16(), FS2:CHECKSUM()

# FILECOPY ()

---

**Syntax:**

**retN = FileCopy ( expC1, expC2, [expL3] )**

**Purpose:**

Copies files

**Arguments:**

<expC1> is the source file name (optionally prefaced by path) that is appended to <expC2>. The file name is used as is, i.e. FS\_SET("path...") is not considered, but you may use TRUEPATH() to convert <expC1> to absolute path.

<expC2> is the designation file name (optionally prefaced by path) to which <expC1> is copied. The file name is used as is, i.e. FS\_SET("path...") is not considered, but you may use TRUEPATH() to convert <expC2> to absolute path.

**Options:**

<expL3> is available for backward compatibility to NT3/CA3 tools only. FlagShip does not support it "backup" mode for floppies.

**Returns:**

<retN> is the number of characters/bytes copied to to <expC2> or -1 on error.

**Description:**

FileCopy() reads the file <expC1> and copies all data to file <expC2>. The file attribute of <expC2> is set to the SetFcreate() rights (default is FA\_RW\_ALL = read/write for all). If CsetSafety() is .T. and the <expC2> file exist, copying is denied.

**Classification:**

add-on library, programming

**Example:**

```
FileCopy("../data/my.dbf", "mysave.dbf")
```

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools, except the 3rd parameter which is ignored in FS2.

**Related:**

FS2:CSETSAFETY(), FS2:SETFCREATE(), FS2:FILEAPPEND(),  
FS2:FILEMOVE(), FS2:FILEDELETE(), FS2:FILEATTR(), FS2:FILESEEK(),  
FS2:FILESIZE(), FS2:FILERIGHTS(), FS2:FILEOWNER(), FS2:FILEGROUP(),  
FS2:FILESYMLNK(), FILE()

# FILEDATE ()

---

**Syntax:**

**retD = FileDate ( [expC1], [expN2] )**

**Purpose:**

Determines the file date

**Arguments:**

<expC1> is the file name (optionally prefaced by path) which access rights has to be determined. The file name is used as is, i.e. FS\_SET("path...") is not considered, but you may use TRUEPATH() to convert <expC1> to absolute path. If <expC1> is not specified, the most-recent file located with FILESEEK() is used.

**Options:**

<expN2> is available for backward compatibility to NT3/CA3 tools only. FlagShip does not support it search by attribute.

**Returns:**

<retD> is the date of last file modification or null-date on error.

**Description:**

FileDate() determines the date of last file modification or of file creation if no changes were done in the meantime. If the <expC1> is a symbolic link, <retD> reports the date of the parent file, not of the symbolic link; an empty date signals dead link.

**Classification:**

add-on library, programming

**Example:**

```
? "File data.dbf was created/changed", ;  
  FileDate("data.dbf"), ;  
  FileTime("data.dbf")
```

**Example:**

see also FS2:SaveFseek() for additional examples

**Compatibility:**

Available in FS2-Toolbox, compatible NT2/CA3 tools, except the 2nd parameter which is ignored in FS2

**Related:**

FS2:FILESEEK(), FS2:FILESIZE(), FS2:FILEATTR(), FS2:FILETIME(),  
FS2:FILEOWNER(), FS2:FILEGROUP(), FS2:FILESYMLNK(), DIRECTORY(),  
FILE(), DATE(), FS2:MDY(), FS2:DMY(), FS2:DOY(), FS2:CDOW2(),  
FS2:CDOM2()

# FILEDELETE ()

---

## **Syntax:**

**retL = FileDelete ( [expC1], [expN2] )**

## **Purpose:**

Deletes file

## **Options:**

<expC1> is the file name, optionally with path, to delete. Wildcards are supported. The file name is used as is, i.e. FS\_SET("path...") and FS\_SET("lower") is not considered, but you may use TRUEPATH() to convert <expC1> to absolute path. If <expC1> is not specified, the file of the most-recent file located with FILESEEK() is used.

<expN2> is available for backward compatibility to NT3/CA3 tools only. FlagShip does not support it search/delete by attribute, you may check the file attribute by using FileAttr() or FileRights()

## **Returns:**

<retL> is .T. on success, i.e. when the file (or at least one file from wildcard) was deleted, and .F. otherwise.

## **Description:**

FileDelete() unrecoverable removes the file <expC1> from hard disk. It is nearly equivalent to FS2:DeleteFile() and differs only in the return value, but FileDelete() can also use wildcards and can remove directories, while DeleteFile() requires single files for the <expC1> argument.

FileDelete() is also similar to standard FlagShip function Ferase() or to the DELETE FILE command; neither Ferase() nor ERASE or DELETE FILE commands supports wildcards.

Note: for security purposes, it is usually better not to remove files blind via wildcard, but determine the wildcard files via FileSeek() and check the file type (e.g. attribute, directory etc) before deleting it with FileDelete() without argument.

If <expC1> does not contain wildcards, DeleteFile() is usually better choice, since it may execute slightly faster and is more secure than FileDelete().

## **Classification:**

add-on library, programming

## **Example:**

```
if FileDelete("../data/my*" + IDEXEXT())
  ? "All indices my*.idx in ../data/ deleted"
endif
INDEX ON ... TO ../data/myIndex
```



**Example:**

```
#include "fileio.fh"
cFile := FileSeek("../data/*")           // first matching file
while len(cFile) > 0
    if BinAnd(FileAttr(), FA_DIRECTORY) == 0 // don't delete dirs
        if !FileDelete() // delete current matching file
            ? "error: could not delete", cFile
        endif
    endif
    cFile := FileSeek() // next matching file
enddo
```

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools, except the 2nd parameter which is ignored in FS2. NT2/CA3 requires the first argument, it does not use the current FileSeek() result.

**Related:**

FS2:FILEAPPEND(), FS2:FILEMOVE(), FS2:FILEATTR(), FS2:FILESEEK(),  
FS2:FILESIZE(), FS2:FILEDATE(), FS2:FILERIGHTS(), FS2:FILEOWNER(),  
FS2:FILEGROUP(), FS2:FILESYMLNK(), FILE()

# FILEGROUP ()

---

**Syntax:**

**retNC = FileGroup ( [expC1], [expL2] )**

**Purpose:**

Determines the group of a file

**Arguments:**

<expC1> is the file name (optionally prefaced by path) which group is to be determined. The file name is used as is, i.e. FS\_SET("path...") and FS\_SET("lower") is not considered, but you may use TRUEPATH() to convert <expC1> to absolute path. If <expC1> is not specified, the most-recent file located with FILESEEK() is used.

**Options:**

<expL2> specifies whether the group name or numeric group ID is returned. If .F. or not given (default), <retN> is the numeric ID of file group. If .T., <retC> is the name of file group.

**Returns:**

<retN> is the numeric ID of file group or -1 on error <retC> with <expL2> = .T. is the name of file group or "" on error

**Description:**

FileGroup() determines the group of the given or last file found. If the <expC1> is a symbolic link, <retNC> reports group of the parent file, not of the symbolic link.

**Classification:**

add-on library, programming

**Example:**

```
cFile := "data.dbf"
? "File", cFile, "(" + ltrim(FileSize(cFile)) + ;
  " bytes) was created/changed", FileDate(cFile), FileTime(cFile)
? "it permission is", FileRights(cFile)
? "the file owner is", FileOwner(cFile, .T.), ;
  "(" + ltrim(FileOwner(cFile)) + ")", ;
  "and the file group is", FileGroup(cFile, .T.), ;
  "(" + ltrim(FileGroup(cFile)) + ")"
```

**Example:**

see also FS2:SaveFseek() for additional examples

**Compatibility:**

Available in FS2-Toolbox, not available in NT2/CA3 tools

**Related:**

FS2:FILEATTR(), FS2:FILESEEK(), FS2:FILEDATE(), FS2:FILETIME(),  
FS2:FILEOWNER(), FS2:FILESYMLNK(), DIRECTORY(), FILE()

# FILEMOVE ()

---

**Syntax:**

**retN = FileMove ( expC1, expC2 )**

**Purpose:**

Moves (or renames) files

**Arguments:**

<expC1> is the source file name (optionally prefaced by path) that is moved to <expC2>. The file name is used as is, i.e. FS\_SET("path...") is not considered, but you may use TRUEPATH() to convert <expC1> to absolute path.

<expC2> is the designation file name (optionally prefaced by path) to which <expC1> is copied. The file name is used as is, i.e. FS\_SET("path...") is not considered, but you may use TRUEPATH() to convert <expC2> to absolute path.

**Returns:**

<retN> is a success or error code:

0	FA_NO_ERROR	No error occurs
-1	FA_FILE_ERROR	Invalid parameter
-2	FA_FILE_NOT_FOUND	File not found
-3	FA_PATH_NOT_FOUND	Path not found (source or designation)
-5	FA_ACCESS_DENIED	Access denied (e.g. the source directory has no search/read access or is not owned by current user or the designation dir has no write/search access) or CsetSafety() is on and <expC2> exists.

where the FA\_\* constants are defined in fileio.fh

**Description:**

FileMove() copies the file <expC1> to <expC2> and on success removes the <expC1>. It is similar to Unix command "mv file2 file1" or the RENAME ... TO... command. If Csetsafety() is set .T., FileMove() is processed only when the designation file <expC2> does not exist.

**Classification:**

add-on library, programming

**Example:**

```
if FileMove("../data/my.dbf", "../save/my.dbf") != 0
  ? "sorry, could not move file 'my.dbf'"
endif
```

**Compatibility:** Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:** FS2:FILEAPPEND(), FS2:FILECOPY(), FS2:FILEDELETE(), FS2:FILEATTR(), FS2:FILERIGHTS(), FS2:FILESEEK(), FS2:FILESIZE(), FS2:FILEOWNER(), FS2:FILEGROUP(), FS2:FILESYMLNK(), FILE()

# FILEOWNER ()

---

## Syntax:

**retNC = FileOwner ( [expC1], [expL2] )**

## Purpose:

Determines the owner of a file

## Arguments:

<expC1> is the file name (optionally prefaced by path) which owner is to be determined. The file name is used as is, i.e. FS\_SET("path...") and FS\_SET("lower") is not considered, but you may use TRUEPATH() to convert <expC1> to absolute path. If <expC1> is not specified, the most-recent file located with FILESEEK() is used.

## Options:

<expL2> specifies whether the owners name or numeric owner ID is returned. If .F. or not given (default), <retN> is the numeric ID of file owner. If .T., <retC> is the name of file owner.

## Returns:

<retN> is the numeric ID of file owner or -1 on error <retC> with <expL2> = .T. is the name of file owner or "" on error

## Description:

FileOwner() determines the owner of the given or last file found. If the <expC1> is a symbolic link, <retNC> reports owner of the parent file, not of the symbolic link.

## Classification:

add-on library, programming

## Example:

```
cFile := "data.dbf"
? "File", cFile, "(" + ltrim(FileSize(cFile)) + ;
  " bytes) was created/changed", FileDate(cFile), FileTime(cFile)
? "it permission is", FileRights(cFile)
? "the file owner is", FileOwner(cFile, .T.), ;
  "(" + ltrim(FileOwner(cFile)) + ")", ;
  "and the file group is", FileGroup(cFile, .T.), ;
  "(" + ltrim(FileGroup(cFile)) + ")"
```

## Example:

see also FS2:SaveFseek() for additional examples

## Compatibility:

Available in FS2-Toolbox, not available in NT2/CA3 tools

## Related:

FS2:FILEATTR(), FS2:FILESEEK(), FS2:FILEDATE(), FS2:FILETIME(),  
FS2:FILEGROUP(), FS2:FILESYMLNK(), DIRECTORY(), FILE()

# FILERIGHTS ()

---

## **Syntax:**

```
retC = FileRights ( expC1, [expNC2], [expL3] )
```

## **Purpose:**

Determine or set file access rights (permission)

## **Arguments:**

<expC1> is the file name (optionally prefaced by path) which access rights has to be determined or changed. Conversion of the file and path to lower/uppercase according to FS\_SET("lower") etc. is done automatically when not disabled by <expL3>

## **Options:**

<expNC2> is either numeric value or a string specifying the new permission (access rights) the file should receive. Accepted only when the file <expC1> is owned by the current user or for root (administrator) user, same as with the "chmod" Unix command.

You either may specify a string in the form "rwxrwxrwx" or a numeric representation, see FS2:FileAttr(). The string is similar to e.g. "ls -la" output; the first "rwx" token represents read, write and execute rights for the file owner, the second "rwx" token for the owner's group and the third "rwx" for others. You may replace any r,w,x character by "-", space or anything else to deny the corresponding permission. For example "rw-rw----" or "rw-rw" gives read/write access for the current user and his group only. A numeric value is accepted too, where e.g. 432 is octal 660 or "rw-rw---" and 436 = octal 664 = "rw-rw-r--", see FILEATTR() or FS2:SETFATTR() for a complete list.

<expL3> If this parameter is .F., the file <expC1> is used "as is". If <expL3> is not given or is .T. (the default), the file and path name in <expC1> is translated to lower/uppercase according to FS\_SET("lower"|"upper"|"path..") or #include "fspreset.fh" setting.

## **Returns:**

<retC> is the <expC1> file permission. When <expNC2> is given, the return value represents the resulting permission. On error, e.g. if the file is not available, or when trying to change permission of file not owned by the current user, or if the directory permission is lower than "rwx", null-string "" is returned.

## **Description:**

FileRights() determines the current access rights of the given file and reports it similarly to "ls -la" output. It also can change the file permission, similarly to "chmod" Unix command. In addition to, the automatic lower/upper case conversion is supported. Note that only the file's owner can change its permission and he must have at least "rwx" access to the directory. The new permission <expNC2> is considered by all consecutive file open attempts for this file.

**Classification:**

add-on library, programming, file manipulation

**Example:**

```
? "test.txt:", filerights("test.txt")           // ""
if !file("test.txt")
    run ("touch test.txt")
    ? "test.txt:", filerights("test.txt")         // rw-r--r--
endif
FS_SET("lower", .T.)
? "test.txt:", filerights("test.txt", "rw")       // rw-----
? "test.txt:", filerights("TEST.TXT", "rw-rw-")  // rw-rw----
? "test.txt:", filerights("test.txt", "rw-rw-rw") // rw-rw-rw-
? "test.txt:", filerights("TEST.txt", , .T.)     // rw-rw--w-
? "test.txt:", filerights("TEST.txt", , .F.)     // ""
? "test.txt:", filerights("test.txt", 6)         // -----rw-
? "test.txt:", filerights("test.txt", 509)       // rwxrwxr-x
wait
```

**Example:**

see also FS2:SaveFseek() for example how to display also links and directory rights

**Compatibility:**

Available in FS2-Toolbox, not available in NT2/CA3 tools.

**Related:**

FS2:FILEATTR(), FS2:FILESTR(), FS2:FILEOWNER(), FS2:FILEGROUP(),  
FS2:FILESYMLNK(), DIRECTORY()

# FILESEEK ()

---

**Syntax:**

**retC = FileSeek ( [expC1], [expN2], [expL3] )**

**Purpose:**

Searches for files

**Arguments:**

<expC1> is the file name (optionally prefaced by path), wildcards are supported. The file name is used as is, i.e. FS\_SET("path...") is not considered, but you may use TRUEPATH() to convert <expC1> to absolute path. This of course apply for files without wildcards.

**Options:**

<expN2> is available for backward compatibility to NT3/CA3 tools only. FlagShip does not support it search by attribute.

<expL3> is available for backward compatibility to NT3/CA3 tools only. FlagShip does not support it search by attribute.

**Returns:**

<retC> is the first or next file found or "" on error.

**Description:**

FileSeek() either searches for the specified file or for the first file matching the wildcard. If <expC1> includes wildcards, <retC> is the first file name found, next matching files are returned by subsequent FILESEEK() invocation without specifying <expC1>. An alternative to FileSeek() is the standard DIRECTORY() function.

**Classification:**

add-on library, programming

**Example:**

```
cFile := FileSeek("*.dbf")    // first matching file
while len(cFile) > 0
    ? "database:", cFile, FileSize(), "bytes"
    cFile := FileSeek()       // next matching file
enddo
```

**Example:**

see also FS2:SaveFseek() for additional examples

**Compatibility:**

Available in FS2-Toolbox, compatible NT2/CA3 tools, except the 2nd and 3rd parameter which is ignored in FS2

**Related:**

FS2:FILESIZE(), FS2:FILEATTR(), FS2:FILEDATE(), FS2:FILETIME(),  
FS2:FILERIGHTS(), FS2:FILEOWNER(), FS2:FILEGROUP(), FS2:FILESYMLNK(),  
DIRECTORY(), FILE()

# FILESIZE ()

---

**Syntax:**

**retN = FileSize ( [expC1], [expN2] )**

**Purpose:**

Determines the size of a file

**Arguments:**

<expC1> is the file name (optionally prefaced by path) which size is to be determined. The file name is used as is, i.e. FS\_SET("path...") is not considered, but you may use TRUEPATH() to convert <expC1> to absolute path. If <expC1> is not specified, the most-recent file located with FILESEEK() is used.

**Options:**

<expN2> is available for backward compatibility to NT3/CA3 tools only. FlagShip does not support it search by attribute.

**Returns:**

<retN> is the file size in bytes or -1 on error.

**Description:**

FileSize() determines the size of the given or last file found. If the <expC1> is a symbolic link, <retN> reports the size of the parent file, not of the symbolic link; -1 signals dead link.

**Classification:**

add-on library, programming

**Example:**

```
cFile := "data.dbf"
? "File", cFile, "(" + ltrim(FileSize(cFile)) + ;
  " bytes) was created/changed", FileDate(cFile), FileTime(cFile)
```

**Example:**

see also FS2:SaveFseek() for additional examples

**Compatibility:**

Available in FS2-Toolbox, compatible NT2/CA3 tools, except the 2nd parameter which is ignored in FS2

**Related:**

FS2:FILESEEK(), FS2:FILEDATE(), FS2:FILETIME(), FS2:FILEOWNER(), FS2:FILEGROUP(), FS2:FILESYMLNK(), DIRECTORY(), FILE()



# FILESTR ()

---

## **Syntax:**

```
retC = FileStr (expC1, [expN2], [expN3], [expL4],  
               [expL5])
```

## **Purpose:**

Reads a portion of a file into a string.

## **Arguments:**

<expC1> The file name from which a string is read. Conversion of the file and path to lower/uppcase according to FS\_SET("lower") etc. is done automatically when not disabled by <expL5>

## **Options:**

<expN2> Specifies how many characters should be read from the file. If <expN2> is not given or is not numeric or is < 0, the default is read all characters (up to 2 Gigabytes or available memory). If the <expN2> is 0, null-string is returned. If <expN1> is larger than the file size, only the bytes available in the file are read and returned.

<expN3> is an offset within the file from which the number of <expN2> bytes (or all bytes/characters up to eof) are read. The default is to read from the first byte (offset = 0) in the file. If <expN3> is larger than the file size, null-string is returned.

<expL4> If this parameter is given as .T., only data up to the first Ctrl-Z (= chr(26)) is read in. The default is read all data (.F.).

<expL5> If this parameter is .F., the file <expC1> is tried to open "as is". If <expL5> is not given or is .T. (the default), the file and path name in <expC1> is translated to lower/uppcase according to FS\_SET("lower"|"upper"|"path..") or #include "fspreset.fh" setting.

## **Returns:**

<retC> is the string read in from the specified file. Embedded zero bytes and binary data are supported. If the file is not available, has not sufficient access rights (at least read-only) or with other errors, null-string "" is silently returned.

## **Description:**

In addition to MEMOREAD() and FREAD(), this FILESTR() offers the capability to read files or a portion of them into a string, providing additional functionality and features over the first.

The counterpart of FILESTR() is STRFILE()

FlagShip supports strings up to 2 Gigabytes, you hence may read up to 2 GB files into memory, presumed you have enough real or virtual memory (swap) available.

**Example:**

```
#include "fspreset.fh" // = FS_SET("lower", .T.)
cNew := "ABCDEFGHIJ" + chr(26) + "KL" + chr(0) + "MN"
STRFILE(cNew, "TEST.TXT", .F.) // create new file

* Read in a file completely:
*
? cVar := FILESTR("TEST.TXT") // ABCDEFGHIJ?KL?MN
// myDisplay(cVar, "cVar")

* Read in everything to the first Ctrl-Z:
*
? cVar := FILESTR("TEST.TXT",,, .T.) // ABCDEFGHIJ
// myDisplay(cVar, "cVar")

* The file TEST.TXT contains "ABCDEFGHIJ...MN". Four characters,
* beginning from position 3 (4th byte), are to be read:
*
? cVar := FILESTR("TEST.TXT", 4, 3) // DEFG
// myDisplay(cVar, "cVar")
wait

function myDisplay(var, txt)
local ii
? procstack(1), txt + " (" + ltrim(len(var)) + ")",var,"="
for ii := 1 to len(var)
?? " " + ltrim(strpeek(var,ii))
next
return NIL
```

**Classification:**

add-on library, programming, file access

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools which supports only the first 4 parameters and file size/strings up to 64KB only.

**Related:**

FS2:STRFILE(), MEMOREAD(), FREAD(), FREADSTR(), FREADTXT()

# FILESYMLNK ()

---

**Syntax:**

**retC = FileSymLnk ( [expC1] )**

**Purpose:**

Determines the parent file name of a symbolic link

**Arguments:**

<expC1> is the file name (optionally prefaced by path) which is a symbolic link. The file name is used as is, i.e. FS\_SET("path...") and FS\_SET("lower") is not considered, but you may use TRUEPATH() to convert <expC1> to absolute path. If <expC1> is not specified, the most-recent file located with FILESEEK() is used.

**Returns:**

<retC> is the parent file name to which <expC1> points, null-string "" signals a dead link or error (i.e. when <expC1> is not symlink).

**Description:**

FileSymLnk() determines the parent file name of a symbolic link.

**Classification:**

add-on library, programming

**Example:**

```
cName := "../data/file.dbf"
if BinAnd(FileAttr(cName), 40960) == 40960
    ? "File", cName, "is a symbolic link to", FileSymLnk(cName)
else
    ? "File", cName, "is not a symbolic link"
endif
```

**Example:**

see also FS2:SaveFseek() for additional examples

**Compatibility:**

Available in FS2-Toolbox, not supported by NT2/CA3 tools.

**Related:**

FS2:FILESEEK(), FS2:FILEATTR(), FS2:FILESIZE(), FS2:FILEDATE(),  
FS2:FILETIME(), FS2:FILEOWNER(), FS2:FILEGROUP(), DIRECTORY(), FILE(),  
TIME(), FS2:TIMETOSEC()

# FILETIME ()

---

**Syntax:**

**retC = FileTime ( [expC1], [expN2] )**

**Purpose:**

Determines the file time

**Arguments:**

<expC1> is the file name (optionally prefaced by path) which access rights has to be determined. The file name is used as is, i.e. FS\_SET("path...") is not considered, but you may use TRUEPATH() to convert <expC1> to absolute path. If <expC1> is not specified, the most-recent file located with FILESEEK() is used.

**Options:**

<expN2> is available for backward compatibility to NT3/CA3 tools only. FlagShip does not support it search by attribute.

**Returns:**

<retC> is the time of last file modification or null string on error. The returned time format is "HH:MM:SS", comparable to TIME() function.

**Description:**

FileTime() determines the time of last file modification or of file creation if no changes were done in the meantime. If the <expC1> is a symbolic link, <retC> reports the time of the parent file, not of the symbolic link; an empty time signals dead link.

**Classification:**

add-on library, programming

**Example:**

```
? "File data.dbf was created/changed", ;  
  FileDate("data.dbf"), ;  
  FileTime("data.dbf")
```

**Example:**

see also FS2:SaveFseek() for additional examples

**Compatibility:**

Available in FS2-Toolbox, compatible NT2/CA3 tools, except the 2nd parameter which is ignored in FS2

**Related:**

FS2:FILESEEK(), FS2:FILESIZE(), FS2:FILEDATE(), DIRECTORY(), FILE(), TIME(), FS2:TIMETOSEC()

# FILEVALID ()

---

**Syntax:**

`retL = FileValid ( expC1 )`

**Purpose:**

Tests whether a string contains valid file name

**Arguments:**

<expC1> is the tested file name, without path and drive.

**Returns:**

<retL> is .T. when the file name <expC1> contains valid characters.

**Description:**

FileValid() only checks if the name <expC1> is valid, i.e. if the file name contain valid characters. <retL> returns .F. when <expC1> contain character(s) other than A..Z, a..z, 0..9, \_ and . or when the string is empty.

FileValid() does not test for the real file availability, where the standard FILE() or the FS2:FILESEEK() functions may be used.

**Classification:**

add-on library, programming

**Example:**

```
cFile := space(20)
@ 1,0 SAY "File name" GET cFile VALID( FileValid( trim(cFile) ) )
READ
```

**Compatibility:**

Available in FS2-Toolbox, compatible NT2/CA3 tools. You may modify the behavior, the function is available in fs2\_udfs.prg source.

**Related:**

FS2:FILESEEK(), FS2:FILESIZE(), FS2:FILEDATE(), DIRECTORY(), FILE(), TIME(), FS2:TIMETOSEC()

# RENAMEFILE ()

---

## Syntax:

**retN = RenameFile ( expC1, expC2 )**

## Purpose:

Fault tolerant renaming of a file

## Arguments:

**<expC1>** is the source file name (optionally prefaced by path) that is renamed to **<expC2>**. The file name is used as is, i.e. FS\_SET("path...") is not considered, but you may use TRUEPATH() to convert **<expC1>** to absolute path.

**<expC2>** is the designation file name (optionally prefaced by path) to which **<expC1>** is copied. The file name is used as is, i.e. FS\_SET("path...") is not considered, but you may use TRUEPATH() to convert **<expC2>** to absolute path.

## Returns:

**<retN>** is a success or error code:

0	FA_NO_ERROR	No error occurs
-1	FA_FILE_ERROR	Invalid parameter
-2	FA_FILE_NOT_FOUND	File not found
-3	FA_PATH_NOT_FOUND	Path not found (source or designation)
-5	FA_ACCESS_DENIED	Access denied (e.g. the source directory has no search/read access or is not owned by current user or the designation dir has no write/search access) or CsetSafety() is on and <b>&lt;expC2&gt;</b> exists.

where the FA\_\* constants are defined in fileio.fh

## Description:

FileRename() renames the file **<expC1>** to **<expC2>**. In fact, it behaves same as FileMove(), i.e. it copies **<expC1>** to **<expC2>** and on success removes the **<expC1>**. It is similar to Unix command "mv file2 file1" or the RENAME ... TO... command. If Csetsafety() is set .T., RenameFile() is processed only when the designation file **<expC2>** does not exist.

## Classification:

add-on library, programming

## Example:

```
if RenameFile("my.dbf", "my.save") != 0
    ? "sorry, could not rename file 'my.dbf' to 'my.save'"
endif
```

**Compatibility:** Available in FS2-Toolbox, compatible to NT2/CA3 tools.

**Related:** FS2:FILEMOVE(), FS2:FILEAPPEND(), FS2:FILECOPY(), FS2:FILEDELETE(), FS2:FILEATTR(), FS2:FLIRIGHTS(), FS2:FILESEEK(), FS2:FILESIZE(), FILE()

# RESTFSEEK ()

---

**Syntax:**

**retL = RestFseek ( expA1 )**

**Purpose:**

Restores the current FILESEEK environment

**Arguments:**

<expA1> is an array containing internal information about FILESEEK() status, stored previously by SaveFseek().

**Returns:**

<retL> is .T. on success and .F. on failure.

**Description:**

The function saves the current FILESEEK() environment to a local variable. It is preferably used for branching in subdirectory and restoring the current status thereafter to continue FileSeek() process.

Note: Always use RestFseek() after performing SaveFseek() to avoid memory leaks.

**Classification:**

add-on library, programming

**Example:**

see FS2:SaveFseek() for complete example

**Compatibility:**

Available in FS2-Toolbox, compatible NT2/CA3 tools, except the passed argument which is character in CA3, which also does not report success or failure.

**Related:**

FS2:SAVEFSEEK(), FS2:FILESEEK(), FS2:FILEATTR(), FS2:FILESIZE(), FS2:FILEDATE(), FS2:FILETIME(), FS2:DIRNAME(), FS2:DIRCHANGE(), DIRECTORY(), FILE()

# SAVEFSEEK ()

---

**Syntax:**

```
retA = SaveFseek ( )
```

**Purpose:**

Saves the current FILESEEK environment

**Returns:**

<retA> is an array containing internal information about FILESEEK() status, required for restoring by RestFseek().

**Description:**

The function saves the current FILESEEK() environment to a local variable. It is preferably used for branching in subdirectory and restoring the current status thereafter to continue FileSeek() process.

Note: Always use RestFseek() after performing SaveFseek() to avoid memory leaks.

**Classification:**

add-on library, programming

**Example:**

```
// list files with corresponding attributes in the current and all
// subdirectories

GetAllFiles("")          // starts with all files in curr directory
// GetAllFiles("abc*")  // starts with abc* files in curr directory
wait

#include "fileio.fh"
* -----
* List all given files in the current directory and all
*   files in sub-directories below.
* Note: instead of the direct output, you may store the
*   current directory, file name and it attributes
*   into array or database and sort it accordingly.
* -----
FUNCTION GetAllFiles(cSearchFiles)
LOCAL cFile, cCurDir
LOCAL aSeekEnv          // stores FILESEEK environment

? "--- directory " + DirName()
? padr(".",20), GetAttrib(".",), str(FileSize("."),10,0), ;
  FileDate(".",), FileTime("."), ;
  FileOwner(".", .T.), FileGroup(".", .T.)

cFile := FileSeek(cSearchFiles) // get first match

WHILE !empty(cFile)
  IF cFile == "." .or. cFile == ".."
    cFile := FileSeek() // Next file
```



```

        loop                                // ".." and "." are ignored
    ENDIF
    IF IsBit(FileAttr(), FA_DIRECTORY) // Subdirectory?
        cCurDir := DirName()             // save current directory
        DirChange(cFile)                  // cd ./<subdirectory>
        IF !(cCurDir == DirName()) // ok?
            aSeekEnv := SaveFseek() // save Fileseek() environment
            GetAllFiles("")           // recursive call (!) to subdir
            RestFseek(aSeekEnv)       // restore Fileseek() environment
            DirChange(cCurDir)       // cd ../
            ? "--- directory " + DirName() + " (continued)"
        ENDIF
    ELSE
        ? padr(cFile, 20), GetAttrib(), str(FileSize(),10,0), ;
        FileDate(), FileTime(), ;
        FileOwner(, .T.), FileGroup(, .T.)
        IF BinAnd(FileAttr(), 40960) == 40960 // symb.link
            ?? " ->", FileSymLnk()
        ENDIF
    ENDIF
    cFile := FileSeek() // Next file
ENDDO
RETURN

* -----
* Creates output similar to FILERIGHTS() but
* including directory or link, e.g. "drwxrwxrwx"
* -----
FUNCTION GetAttrib(cPath)
local iAttr, cRet
if cPath == NIL
    iAttr := FileAttr()
else
    iAttr := FileAttr(cPath)
endif
if iAttr == 0
    return space(10)
endif

if IsBit(iAttr, 15)
    cRet := "d" // directory
elseif IsBit(iAttr, 16) .and. IsBit(iAttr, 14)
    cRet := "l" // symb. link
elseif IsBit(iAttr, 16)
    cRet := "-" // regular file
else
    cRet := "?" // other
endif

cRet += if(IsBit(iAttr, 9), "r", "-") // owner
cRet += if(IsBit(iAttr, 8), "w", "-")
cRet += if(IsBit(iAttr, 7), "x", "-")
cRet += if(IsBit(iAttr, 6), "r", "-") // group
cRet += if(IsBit(iAttr, 5), "w", "-")
cRet += if(IsBit(iAttr, 4), "x", "-")
cRet += if(IsBit(iAttr, 3), "r", "-") // others
cRet += if(IsBit(iAttr, 2), "w", "-")

```

```
cRet += if(IsBit(iAttr, 2), "x","-")  
return cRet
```

***Compatibility:***

Available in FS2-Toolbox, compatible NT2/CA3 tools, except the return value and its type which is character in CA3

***Related:***

FS2:RESTFSEEK(), FS2:FILESEEK(), FS2:FILEATTR(), FS2:FILESIZE(),  
FS2:FILEDATE(), FS2:FILETIME(), FS2:DIRNAME(), FS2:DIRCHANGE(),  
DIRECTORY(), FILE()

# SETFATTR ()

---

## Syntax:

```
retC = SetFattr ( [expC1], [expN2] )
```

## Purpose:

Sets a file's attributes - subset of FileRights()

## Options:

<expC1> is the file name (optionally prefaced by path) which access rights (attributes) has to be changed. The file name is used as is, i.e. FS\_SET("path...") and FS\_SET("lower") is not considered, but you may use TRUEPATH() to convert <expC1> to absolute path. If <expC1> is not specified, the most-recent file located with FILESEEK() is used.

<expN2> is numeric value specifying the new permission (access rights) the file should receive. Accepted only when the file <expC1> is owned by the current user, or for root (administrator) user. If <expN2> is not specified or is NIL, FA\_RW\_ALL constant is used.

Following Unix attributes are accepted:

Value	octal	Constant	Assigned attribute
2048	0004000		set UID bit (use with care!)
1024	0002000		set GID bit (use with care!)
512	0001000		sticky bit (use with care!)
493	0000755	FA_EXEC_ALL	execute for all = rwxr-xr-x
438	0000666	FA_RW_ALL	read/write for all = rw-rw-rw-
420	0000644	FA_NORMAL	read/write owner = rw-r--r--
256	0000400	FA_READONLY	owner has read permission
128	0000200	FA_WRITEONLY	owner has write permission
64	0000100	FA_EXECUTE	owner has execute permission
32	0000040		group has read permission
16	0000020		group has write permission
8	0000010		group has execute permission
4	0000004		others have read permission
2	0000002		others have write permission
1	0000001		others have execute permission
0		FA_HIDDEN	HIDDEN n/a on Unix
0		FA_SYSTEM	SYSTEM n/a on Unix
0		FA_VOLUME	VOLUME n/a on Unix
0		FA_ARCHIVE	ARCHIVE n/a on Unix

In MS-DOS, following attributes are accepted:

Value	octal	Constant	Assigned attribute
0	0000000	FA_NORMAL	Normal
0	0000000	FA_RW_ALL	Normal
0	0000000	FA_EXEC_ALL	Normal
1	0000001	FA_READONLY	READ ONLY (Read-only)
2	0000002	FA_HIDDEN	HIDDEN (Hidden files)
4	0000004	FA_SYSTEM	SYSTEM (System files)
8	0000010	FA_VOLUME	VOLUME (Name of a floppy/hard disk)
16	0000020	FA_DIRECTORY	DIR (Directory)
32	0000040	FA_ARCHIVE	ARCHIVE (Changes since last backup)

where the required file attributes should be added to each other. For cross-compatible application to Unix and MS-Windows (or DOS), the symbolic constants, defined in fileio.fh, should be preferred.

### Returns:

<retN> is a success or error code:

0	FA_NO_ERROR	No error occurs
-1	FA_FILE_ERROR	Invalid parameter
-2	FA_FILE_NOT_FOUND	File not found
-3	FA_PATH_NOT_FOUND	Path not found (source or designation)
-5	FA_ACCESS_DENIED	Access denied (e.g. the source directory has no search/read access)

where the FA\_\* constants are defined in fileio.fh

### Description:

SetFattr() is NT2/CA3 compatible subset of the FS2:FileRights() function which should be preferably used. Note that only the file's owner (or root/administrator) can change it permission and he must have at least "rwx" access to the directory. The new permission <expN2> is considered by all consecutive file open attempts for this file.

### Classification:

add-on library, programming, file manipulation

### Example:

```
if SetFattr( TruePath("Test.TXT"), FA_READONLY) != FA_NO_ERROR
    alert("could not set attribute for Test.TXT")
endif
```

### Example:

see also FS2:FileRights() for more examples

**Compatibility:** Available in FS2-Toolbox, compatible NT2/CA3 tools.

**Related:** FS2:FILERIGHTS(), FS2:SETFDAT(), FS2:FILEATTR(), FS2:FILEOWNER(), FS2:FILEGROUP(), FS2:FILESYMLNK(), DIRECTORY()

# SETFCREATE ()

---

**Syntax:**

**retN = SetFcreate ( [expN1] )**

**Purpose:**

Sets default file attribute for creating with FS2 functions

**Options:**

<expN1> is the default file attribute (see FS2:SetFAttr() for a list) used when a new file is created by using FS2 functions. If <expN1> is not specified, only the current attribute is returned which is the FA\_RW\_ALL + FA\_ARCHIVE constant if not re-defined otherwise.

**Returns:**

<retN> is the old SetFcreate() attribute, i.e. the current one at the time of entering this function. If <retN> is equivalent to <expN1>, the attribute could not be set (invalid attribute).

**Description:**

SetFcreate() attribute is considered when creating new file by using FS2:FileCopy(), FS2:FileAppend() and FS2:StrFile() - but not in FS2:FileRename(), FS2:FileMove() nor in standard commands/functions, where the attributes can be set by FS2:SetFattr() or FS2:FileRights() thereafter.

**Example:**

```
#include "fileio.fh"
iNewFlag := FA_NORMAL           // value differs for DOS and Unix
iOldFlag := SetFcreate(iNewFlag)

if iOldFlag != iNewFlag .and. SetFcreate() == iOldFlag
    ? "Sorry, wrong parameter:", ltrim(iNewFlag),"is out of range"
elseif iOldFlag == iNewFlag
    ? "SetFcreate flag remain unchanged:", ltrim(SetFcreate())
else
    ? "SetFcreate flag changed from", ltrim(iOldFlag), "to", ;
    ltrim(SetFcreate()), ;
    "and will be considered for next FILE*() operation"
endif
```

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools. Note the different constant FA\_\* values (see SetFattr() and fileio.fh) for application running in DOS and Unix

**Related:**

FS2:FILECOPY(), FS2:FILEAPPEND(), FS2:STRFILE(), FS2:FILERIGHTS(),  
FS2:FILEATTR(), FS2:SETFATTR(), FS2:SETFDATI(), FS2:FILEOWNER(),  
FS2:FILEGROUP(), FS2:FILESYMLNK(), FS2:FILEDATE(), FS2:FILETIME(),  
DIRECTORY()

# SETFDATI ()

---

## Syntax:

```
retL = SetFdati ( [expC1], [expD2], [expC3] )
```

## Purpose:

Sets the date and/or time of a file or directory

## Options:

<expC1> is the file name (optionally prefaced by path) which access rights (attributes) has to be changed. The file name is used as is, i.e. FS\_SET("path...") and FS\_SET("lower") is not considered, but you may use TRUEPATH() to convert <expC1> to absolute path. If <expC1> is not specified, the most-recent file located with FILESEEK() is used.

<expD2> is the new date to be set for the file, using e.g. DATE(), CTOD() etc. If not specified or is NIL, current system DATE() is used. If the date should remain unchanged, pass .F. or FileDate(<expC1>)

<expC3> is the new time to be set for the file, given in "HH:MM:SS" format, same as TIME() output, "HH:MM" is accepted as well. If not specified or is NIL, current system TIME() is used. If the time should remain unchanged, pass .F. or FileTime(<expC1>)

## Returns:

<retL> is .T. on success; .F. reports error, e.g. if the date or time could not be set since the file does not exist or the current user has not enough access rights, or on invalid parameters.

## Description:

SETFDATI() set the modification date and/or time of a file or directory given in <expC1>

## Example:

```
#include "fspreset.fh"
cFile := TruePath("../data/MyFile.data")

? "The date and time of the file " + cFile + ;
  "was changed from", FileDate(cFile), FileTime(cFile)
SetFdati(cFile, date() +1, "13:55:59")
?? " to", FileDate(cFile), FileTime(cFile)
```

## Example:

```
cFile := "test.dat"
if !file(cFile)
  StrFile("hello", cFile)
endif

? "File", cFile, FileDate(cFile), FileTime(cFile)
?? " -> date 11/30/02 = "
?? SetFdati(cFile, ctod("11/30/02"), .F.) // change date only
?? "", FileDate(cFile), FileTime(cFile)
```

```
? "File", cFile, FileDate(cFile), FileTime(cFile)
?? " -> time 13:15:05 = "
?? SetFdati(cFile, .F., "13:15:05")           // change time only
?? "", FileDate(cFile), FileTime(cFile)

? "File", cFile, FileDate(cFile), FileTime(cFile)
?? " -> now = "
?? SetFdati(cFile)                          // set current date & time
?? "", FileDate(cFile), FileTime(cFile)
```

**Compatibility:**

Available in FS2-Toolbox, compatible NT2/CA3 tools.

**Related:**

FS2:FILEDATE(), FS2:FILETIME(), FS2:FILERIGHTS(), FS2:SETFATTR(),  
FS2:FILEATTR(), FS2:FILEOWNER(), FS2:FILEGROUP(), FS2:FILESYMLNK(),  
DIRECTORY(), DATE(), TIME()

# STRFILE ()

---

## Syntax:

```
retN = StrFile ( expC1, expC2, [expL3], [expN4],  
                [expL5], [expC6], [expL7], [expNC8] )
```

## Purpose:

Writes a string to a file.

## Arguments:

<expC1> is the string to write to file <expC2>. Embedded zero bytes are supported.

<expC2> is the file name to be created or overwritten. Conversion of the file and path to lower/uppercase according to FS\_SET("lower") etc. is done automatically when not disabled by <expL7>

## Options:

<expL3> if an optional flag specifying whether an existing file should be overwritten (.T.) or truncated / newly created (.F.). The default is .F. Of course, when the file does not exist, it is always newly created.

<expN4> is an offset (starting with 0) within the file from which the string <expC1> is to be written. When <expN4> is not given or is < 0, <expC1> string is appended to end of file. When the <expN4> offset is greater than the current file size, the hole (between eof and <expN4>) is filled by the <expC6> character.

<expL5> if an optional flag specifying whether an existing file should be truncated (.T.) behind the last character of <expC1>. The default is .F. means the original file size remains unchanged when <expC1> overwrites an available part of it, provided that <expL3> is .T.

<expC6> is an optional character used to fill a hole between end-of- file and the byte offset <expN4>. If not specified, chr(0) is used. If the string is longer than 1, only it first character is used.

<expL7> If this parameter is .F., the file <expC2> is tried to open "as is". If <expL7> is not given or is .T. (the default), the file and path name in <expC2> is translated to lower/uppercase according to FS\_SET("lower"|"upper"|"path..") or #include "fspreset.fh" setting.

<expNC8> is either a numeric value or a string specifying permission (access rights) of the file <expC1>. The string is in the form "rwxrwxrwx", the numeric value represents it octal value, see further details in FS2:FLERIGHTS() and FILEATTR(). If not specified, a new file is created with default permission according to SetFcreate() permission (which is usually FA\_RW\_ALL + FA\_ARCHIVE = rw-rw-rw- on Unix and 32 on DOS); the permission of available file remain unchanged. Note: you may change the file access rights only when you are owner of this file. The permission <expNC8> is set after the file is created or changed, it hence does not affect the current StrFile() operation nor already open files, but influences all read/write operations to this file opened thereafter.



**Returns:**

<retN> reports the number of bytes of <expC1> written into the file <expC2>. On error (e.g. on insufficient file or directory permission), 0 is silently reported.

**Description:**

In addition to MEMOWRITE() and FWRITE(), this STRFILE() offers the capability to write ascii or binary data into a file. STRFILE() offers addition features and extended functionality.

To be able to overwrite existing file, you need at least "rw" access right to it, and "rwx" access in the directory. The last is required also to create new file.

The counterpart of STRFILE() is FILESTR()

FlagShip supports strings up to 2 Gigabytes, hence you may write up to 2 GB files from memory into the file; it only presume you have enough real or virtual memory (swap) available.

**Example:**

```
#include "fspreset.fh"                // = FS_SET("lower", .T.)
cVar := "ABCDEFGHIJ" + chr(26) + "KLM"

? STRFILE(cVar, "TEST.TXT",,,,,,"rw") // 14
// cVar := FILESTR("TEST.TXT")        // ABCDEFGHIJ?KLM
// myDisplay(cVar, "cVar")

* Add to the end of a file:
*
? STRFILE("123456", "TEST.TXT", .T.,,,,,,"rw") // 6
// cVar := FILESTR("TEST.TXT")          // ABCDEFGHIJ?KLM123456
// myDisplay(cVar, "cVar")

* Data in an existing file is overwritten from offset 4
* (i.e. 5th byte) with a designated string "FlagShip"
*
? STRFILE("FlagShip", "TEST.TXT", .T., 4) // Result: 8
// cVar := FILESTR("TEST.TXT")           // ABCDFlagShipLM123456
// myDisplay(cVar, "cVar")
wait

FUNCTION myDisplay(var, txt)
local ii
? procstack(1), txt + " (" + ltrim(len(var)) + ")",var,"="
for ii := 1 to len(var)
?? " " + ltrim(strpeek(var,ii))
next
return NIL
```

### Example:

```
// DELETE FILE test.txt
// DELETE FILE TEST.TXT

FS_SET("lower", .T.) // translate to lower

* Create new file named "test.txt" with 15 characters
* Note the automatic upper/lower conversion:
*
? STRFILE(REPLICATE("x", 15), "TEST.TXT") // 15
// cVar := FILESTR("test.txt") // xxxxxxxxxxxxxxxxx
// myDisplay(cVar, "(#1) cVar")

* A 5-character string is written starting at offset 6 (7th byte)
* in an existing file 15-characters long. Since the final parameter
* is specified as .T. once, and specified as .F. once, you see
* different results:
*
? STRFILE("AAAAA", "TEST.TXT", .T., 6, .F.) // 5
// cVar := FILESTR("TEST.TXT") // xxxxxxAAAAAxxxx
// myDisplay(cVar, "(#2) cVar")

? STRFILE("BBB", "TEST.TXT", .T., 9, .T.,, "r") // 3
// cVar := FILESTR("TEST.TXT") // xxxxxxAAABBB
// myDisplay(cVar, "(#3) cVar")
? FILERIGHTS("test.txt") // r-----

* The file "test.txt" was previously set to read-only
* and has now insufficient access rights to write
*
? STRFILE("CCC", "test.txt", .T., 3) // 0
// cVar := FILESTR("test.txt") // xxxxxxAAABBB
// myDisplay(cVar, "(#4) cVar")

? FILERIGHTS("test.txt", "rw-rw") // rw-rw----

* The file is now 12 bytes long. Writing 3 characters
* at offset 14 (i.e. 15th byte) will cause a hole
* filled here by spaces
*
? STRFILE("CCC", "test.txt", .T., 14,, " ") // 3
// cVar := FILESTR("test.txt") // xxxxxxAAABBB CCC
// myDisplay(cVar, "(#5) cVar")

* The file "TEST.TXT" is now newly created, since
* used "as is" w/o automatic upper/lower conversion,
* and 3 chars are written starting at offset 3,
* the hole is filled by chr(0)
*
? STRFILE("DDD", "TEST.TXT", .T., 3,,, .F.) // 3
// cVar := FILESTR("TEST.TXT",,,, .F.) // ???DDD
// myDisplay(cVar, "(#6) cVar")
? FILERIGHTS("TEST.TXT", , .F.) // rw-r--r--
wait
```

**Classification:**

add-on library, programming, file access

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools, where CA Tools supports only the first 5 arguments and strings of max 64 KB.

**Related:**

FS2:FILESTR(), FS2:SETFCREATE(), FS2:FILEATTR(), FS2:FILERIGHTS(),  
FS2:SETFATTR(), FS2:SETFDATI(), MEMOWRITE(), FWRITE()

# TEMPFILE ()

---

## Syntax:

```
retC = TempFile ( [expC1], [expC2], [expNC3],  
                  [expC4] )
```

## Purpose:

Creates a file for temporary use

## Options:

**<expC1>** is optional drive and/or directory where the temporary file is to be created. For drives, an automatic translation/mapping according to ?\_FSDRIVE (see LNG.9.5 and FSC.3.3) is performed, but not the full upper/lower translation via FS\_SET("path..") etc. If **<expC1>** is not specified, the default is the current drive/directory. Note, the environment variable TMPDIR will override this parameter.

**<expC2>** is optional file extension (with or without period). If not specified, the created temp file is without extension.

**<expNC3>** is either numeric value or a string specifying the new permission (access rights) the temp file should receive. See FILERIGHTS() or FILEATTR() for the character or numeric values.

**<expC4>** is a favorite file name initial letter(s). If not given, or specified null-string or NIL, a fully random name is generated.

## Returns:

**<retC>** is the generated, random and unique file name, with or without extension. On error, null-string is returned.

## Description:

This function generates a new, empty file that can be safely used for temporary files. The difference between standard TempFileName() and this TempFile() function is mainly in creation of the file. The algorithm and returned string is similar (but not equivalent) to the system function tempnam(), i.e.

```
[<path>]/[<prefix>]<PID><postfix>[.<ext>]
```

where the

**<path>** is either **<expC1>** or ./ or /tmp or TMPDIR envir.var

**<prefix>** are up to the first 5 characters of **<expC4>**

**<PID>** is the process ID number, last 5 digits

**<postfix>** are three random characters

**<ext>** is **<expC2>** if specified

If the file could not be created in the given directory, either TMPDIR or /tmp directory is used instead. If all trial fails, null string is returned.

## Example:

```
#include "fileio.fh"  
? TempFile("./") // /home/29525paa  
? TempFile("./test", , , "abcdefgh") // /home/test/abcde29525qaa  
? TempFile("./not_avail/", "temp", ;  
  FA_NORMAL, "abcdefgh") // /tmp/abcde29525raa.temp
```

**Classification:**

add-on library, programming, file access

**Compatibility:**

Available in FS2-Toolbox, compatible to NT2/CA3 tools, where CA Tools supports only the first 3 arguments.

**Related:**

TEMPFILENAME(), FS2:FILESTR(), FS2:SETFCREATE(), FS2:FILEATTR(),  
FS2:FILERIGHTS(), FS2:SETFATTR(), FS2:SETFDATI(), FS2:STRFILE(),  
MEMOWRITE(), FWRITE()

# TRUENAME ()

---

**Syntax:**

**retC = TRUENAME (expC1)**

**Purpose:**

Determines the true path or the full file name including a path.

**Arguments:**

<expC1> is either a relative path to be converted to an absolute one, or a file name of which the full path is to be determined the same way as the FILE() function searches for it. If only a path is specified, the string has to be terminated by the "\" or "/" character.

**Returns:**

<retC> is the expanded absolute path, optionally including the file name. If the file is not found, a null string "" is returned. If only a path is specified, no check for its existence is performed.

**Description:**

TRUENAME() is equivalent to standard FlagShip function TRUEPATH(). It determines the absolute UNIX or DOS path. If a path only is specified, the FS\_SET() translations and the drive substitution (using the environment variable x\_FSDRIVE) is considered. If a file name is given as well, the current SET PATH is also considered.

**Example:**

```
? CURDIR ()                && /usr/john/tmp
? TRUEPATH ("./")          && /usr/john/tmp/
? TRUENAME ("./")          && /usr/john/tmp/
? TRUEPATH ("..\..\data\") && /usr/data/
? TRUENAME ("..\..\data\") && /usr/data/

FS_SET ("lower, .T.)
? TRUEPATH ("Adr.DBF")      && /usr/john/adr/adr.dbf
? TRUENAME ("Adr.DBF")     && /usr/john/adr/adr.dbf
```

**Example:**

See further examples in TRUEPATH()

**Classification:**

programming

**Compatibility:**

Available in FS2-Toolbox, compatible NT2/CA3 tools.

**Related:**

TRUEPATH(), FindExeFile(), FS\_SET(), FILE(), SET PATH, x\_FSDRIVE

# Settings, System Functions

---

## Introduction

With the FS2 Toolbox Setting and System functions, you can change the behavior of some FS2 Toolbox functions, set or receive system status or (partially) manipulate the system self.

The Extended Get/Read functions provide additional information and handling of standard FlagShip @..GET and READ commands.

## Index of Setting and System Functions

BLANK()	Creates a blank value for each data type - see FS2 String
* BIOSDATE()	Determines the system BIOS date
* BOOTCOLD()	Triggers a cold boot
* BOOTWARM()	Triggers a warm start of the system
COMPLEMENT()	Forms the complement value of a data type - see FS2 String
* CPUTYPE()	Determines what type of microprocessor in use
CSETALL()	Saves all ON/OFF switch settings
CSETCLIP()	Determines the content of environmental variables
CSETDATE()	Queries the SET DATE setting
CSETDECI()	Queries the setting for SET DECIMALS TO
CSETDEFA()	Queries the setting for SET DEFAULT
CSETFUNC()	Queries the setting for SET FUNCTION TO
CSETKEY()	Queries the setting for SET KEY TO
CSETLDEL()	Queries the setting for the left delimiters
CSETMARG()	Queries the setting for SET MARGIN TO
CSETPATH()	Queries the setting for SET PATH TO
CSETRDEL()	Queries the setting for the right delimiter
CSETRDELI()	equivalent to CSETRDEL()
CSETRDONLY()	Queries/sets the read-only mode switch
CSETSAFETY()	Queries/sets the safety mode switch
CSETSNOW()	Helps prevent snow on the screen
CSETxxx()	Queries an ON/OFF switch and optionally sets it
DATATYPE()	Determines the data type of a variable or UDF
DOSPARAM()	Retrieves the command line parameters as a string
ENVPARAM()	Reads the entire OS environment table into a string
* ERRORACT()	Recommends action for a DOS error occurred previously
* ERRORBASE()	Source of the most-recent DOS error
* ERRORCODE()	Identifies a DOS error that has occurred previously
* ERRORORG()	Origin of the most-recent DOS error
EXENAME()	Returns name and directory of the current program
FILESFREE()	Specifies the number of files you can open (fs2_udfs.prg)
FILESMAX()	Specifies maximum number of open files (fs2_udfs.prg)

* GETCOUNTRY()	Queries country setting for the operating system
GETTIC()	Determines the number of timer ticks (fs2_udfs.prg)
* INBYTE()	Reads an 8 byte from a port
* INWORD()	Reads in a 16-bit word from a port
* ISANSI()	Tests to see if the ANSI screen driver is installed
* ISAT()	Determines if a program is running on an AT
ISDEBUG()	Determines if the debugger is available in the application
* ISMATH()	Determines if a math coprocessor is installed
* KBDDISABLE()	Locks/unlocks the keyboard
KBDEMULATE()	Inserts characters into BIOS keyboard buffer (fs2_udfs.prg)
* KBDSPPEED()	Sets keyboard auto repeat speed
* KBDSTAT()	Tests for key shift state status, such as Ctrl and Shift
* KBDTYPE()	Determines the type of keyboard in use
KEYSEC()	Triggers a key trap after a time delay - see FS2:Time funct.
KEYTIME()	Triggers a key trap at a specific clock time - see FS2:Time
* KSETCAPS()	Queries/sets the system setting for CAPS LOCK
KSETINS()	Queries/sets the system setting for "INSERT"
* KSETNUM()	Queries/sets the system setting for NUMLOCK
* KSETSCROLL()	Queries/sets the system setting for SCROLL LOCK
LASTKFUNC()	Returns the trap function that created the most-recent trap
LASTKLINE()	Returns the line number of the most-recent key trap
LASTKPROC()	Returns the procedure name that was interrupted
MILLISEC()	Time delay in milliseconds
* MEMSIZE()	Determines size of conventional or extended memory
* NUMFKEY()	Determines the number of function keys
NUL()	Converts the value returned by a function into a null string
* NUMBUFFERS()	Determines the BUFFERS= setting in DOS CONFIG.SYS
* NUMFILES()	Determines the FILES= in DOS CONFIG.SYS = FILESMAX()
* OSVER()	Returns the DOS version number
* OUTBYTE()	Sends a byte to a port
* OUTWORD()	Sends a 16-bit word to a port
* PCTYPE()	Returns the type of computer in use
* PEEKBYTE()	Reads a byte from memory
* PEEKSTR()	Reads a byte sequence from memory
* PEEKWORD()	Reads a 16-bit word from memory
* POKEBYTE()	Writes a byte to memory
* POKEWORD()	Writes a 16-bit word to memory
* SCANKEY()	Queries scan code of keyboard input
SETLASTKEY()	Sets the value for LASTKEY()
SETTIC()	Increases number of time ticks (fs2_udfs.prg)
* SHOWKEY()	Continuously displays the INSERT and LOCK status
* SOUND()	Creates tones by designating frequency and duration
* SPEED()	A comparison value of the processor speed to 4.77 MHz PC <g>
* SSETBREAK()	Sets and checks the DOS BREAK switch
* SSETVERIFY()	Sets and checks the DOS VERIFY switch
* STACKFREE()	Determines the remaining stack space
TOOLVER()	Queries the version number of the FS2 Tools in use



XTOC()                      Converts an expression of any data type into a string

### **Index of Get/Read extensions**

COUNTGETS()	Determines the number of posted GET fields
CURRENTGET()	Determines the number of the currently active GET field
GETFLDCOL()	Determines the screen column of a GET field
GETFLDROW()	Determines the row of a GET field on the screen
GETFLDVAR()	Determines the name of a GET field
GETINPUT()	Keyboard input function similar to a GET field (fs2_udfs.prg)
GETSECRET()	Keyboard input function for hidden input (fs2_udfs.prg)
RESTGETS()	Restores GET settings from an array
RESTSETKEY()	Restores SET KEY..TO settings from an array
SAVEGETS()	Saves the GET settings of the active environment
SAVESETKEY()	Saves SET KEY..TO settings in an array

Functions marked by (\*) are available for compatibility purposes only in the source file fs2\_udfs.prg. When FS\_SET("devel",.T.) is set, you will receive developer's warning, otherwise default return value only. These functions are not applicable in Unix nor are cross-portable and



# Index

---

---

## A

Acos( ) ..... FS2-140  
AddAscii( ) ..... FS2-11  
AddMonth( ) ..... FS2-197  
AfterAtNum( ) ..... FS2-12  
AsciiSum( ) ..... FS2-14  
AscPos( ) ..... FS2-15  
Asin( ) ..... FS2-141  
AtAdjust( ) ..... FS2-16  
Atan( ) ..... FS2-142  
Atn2( ) ..... FS2-143  
AtNum( ) ..... FS2-18  
AtRepl( ) ..... FS2-20  
AtToken( ) ..... FS2-22

---

## B

BeforAtNum( ) ..... FS2-24  
BiosDate( ) ..... FS2-363  
BitToC( ) ..... FS2-164  
Blank( ) ..... FS2-26  
BoM( ) ..... FS2-198  
BootCold( ) ..... FS2-363  
BootWarm( ) ..... FS2-363  
BoQ( ) ..... FS2-199  
BoY( ) ..... FS2-200

---

## C

Cdow2( ) ..... FS2-201  
Ceiling( ) ..... FS2-144  
Celsius( ) ..... FS2-165  
Center( ) ..... FS2-27  
Cga40( ) ..... FS2-283  
Cga80( ) ..... FS2-283  
CharAdd( ) ..... FS2-29  
CharAnd( ) ..... FS2-30  
CharEven( ) ..... FS2-31  
CharList( ) ..... FS2-32  
CharMirr( ) ..... FS2-33  
CharMix( ) ..... FS2-34  
CharNolist( ) ..... FS2-35  
CharNot( ) ..... FS2-36

CharOdd( ) ..... FS2-37  
CharOne( ) ..... FS2-38  
CharOnly( ) ..... FS2-39  
CharOr( ) ..... FS2-40  
CharPack( ) ..... FS2-42  
CharPix( ) ..... FS2-283  
CharRela( ) ..... FS2-44  
CharRelRep( ) ..... FS2-45  
CharRem( ) ..... FS2-46  
CharRepl( ) ..... FS2-47  
CharSort( ) ..... FS2-49  
CharSpread( ) ..... FS2-51  
CharSwap( ) ..... FS2-52  
CharUnpack( ) ..... FS2-53  
CharWin( ) ..... FS2-283  
CharXor( ) ..... FS2-54  
Checksum( ) ..... FS2-56  
ClearBit( ) ..... FS2-166  
ClearEol( ) ..... FS2-285  
ClearSlow( ) ..... FS2-283  
ClearWin( ) ..... FS2-287  
CIeol( ) ..... FS2-289  
CIWin( ) ..... FS2-290  
Cmonth2( ) ..... FS2-202  
Color  
    - convert  
        -- from numeric ..... FS2-300  
        -- to numeric ..... FS2-291  
    - select  
        -- enhanced ..... FS2-293  
ColorRepl( ) ..... FS2-283  
ColorToN( ) ..... FS2-291  
ColorWin( ) ..... FS2-283  
COM\_BREAK( ) ..... FS2-310  
COM\_CLOSE( ) ..... FS2-310  
COM\_COUNT( ) ..... FS2-310  
COM\_CRC( ) ..... FS2-310  
COM\_CTS( ) ..... FS2-310  
COM\_DCD( ) ..... FS2-310  
COM\_DOSCON( ) ..... FS2-310  
COM\_DSR( ) ..... FS2-310  
COM\_DTR( ) ..... FS2-310  
COM\_ERRCHR( ) ..... FS2-310  
COM\_ERROR( ) ..... FS2-310  
COM\_EVENT( ) ..... FS2-310

COM\_FLUSH( ) ..... FS2-310  
 COM\_GETIO( ) ..... FS2-310  
 COM\_GETIRQ( ) ..... FS2-310  
 COM\_HARD( ) ..... FS2-310  
 COM\_INIT( ) ..... FS2-310  
 COM\_INIT2( ) ..... FS2-310  
 COM\_ISOPEN( ) ..... FS2-310  
 COM\_KEY( ) ..... FS2-310  
 COM\_LSR( ) ..... FS2-310  
 COM\_MCR( ) ..... FS2-310  
 COM\_MSR( ) ..... FS2-310  
 COM\_NUM( ) ..... FS2-310  
 COM\_OPEN( ) ..... FS2-310  
 COM\_READ( ) ..... FS2-310  
 COM\_REMOTE( ) ..... FS2-310  
 COM\_RING( ) ..... FS2-310  
 COM\_RTS( ) ..... FS2-310  
 COM\_SCOUNT( ) ..... FS2-310  
 COM\_SEND( ) ..... FS2-310  
 COM\_SETIO( ) ..... FS2-310  
 COM\_SETIRQ( ) ..... FS2-310  
 COM\_SFLUSH( ) ..... FS2-310  
 COM\_SKEY( ) ..... FS2-310  
 COM\_SMODE( ) ..... FS2-310  
 COM\_SOFT( ) ..... FS2-310  
 COM\_SOFT\_R( ) ..... FS2-310  
 COM\_SOFT\_S( ) ..... FS2-310  
 Complement( ) ..... FS2-58, 363  
 Cos( ) ..... FS2-145  
 Cot( ) ..... FS2-146  
 CountGets( ) ..... FS2-365  
 CountLeft( ) ..... FS2-59  
 CountRight( ) ..... FS2-60  
 Cputype( ) ..... FS2-363  
 Crc16( ) ..... FS2-61  
 Crypt( ) ..... FS2-63  
 CsetAll( ) ..... FS2-363  
 CsetAtMupa( ) ..... FS2-65  
 CsetClip( ) ..... FS2-363  
 CsetDate( ) ..... FS2-363  
 CsetDeci( ) ..... FS2-363  
 CsetDefa( ) ..... FS2-363  
 CsetFunc( ) ..... FS2-363  
 CsetKey( ) ..... FS2-363  
 CsetLdel( ) ..... FS2-363  
 CsetMarg( ) ..... FS2-363  
 CsetPath( ) ..... FS2-363  
 CsetRdel( ) ..... FS2-363  
 CsetRdeli( ) ..... FS2-363

CsetRdonly( ) ..... FS2-363  
 CsetRef( ) ..... FS2-66  
 CsetSafety( ) ..... FS2-363  
 CsetSnow( ) ..... FS2-363  
 Csetxxxx( ) ..... FS2-363  
 CtoBit( ) ..... FS2-167  
 CtoDow( ) ..... FS2-203  
 CtoF( ) ..... FS2-168  
 CtoMonth( ) ..... FS2-204  
 CtoN( ) ..... FS2-169  
 CurrentGet( ) ..... FS2-365

---

## D

DataType( ) ..... FS2-363  
 Date  
   - complement ..... FS2-58  
   - convert  
     -- character month ..... FS2-202  
     -- character weekday ..... FS2-201  
     -- to string ..... FS2-205, 218  
   - day  
     -- first of month ..... FS2-198  
     -- first of quarter ..... FS2-199  
     -- first of year ..... FS2-200  
     -- last of month ..... FS2-208  
     -- last of quarter ..... FS2-209  
     -- last of year ..... FS2-210  
     -- of year ..... FS2-207  
   - month  
     -- add ..... FS2-197  
     -- character ..... FS2-202  
     -- first day of ..... FS2-198  
     -- last day of ..... FS2-208  
     -- name ..... FS2-221  
     -- number of days ..... FS2-217  
     -- subtract ..... FS2-197  
   - name of day ..... FS2-203  
   - name of month ..... FS2-204  
   - quarter  
     -- first day of ..... FS2-199  
     -- last day of ..... FS2-209  
     -- number of ..... FS2-222  
   - set ..... FS2-225  
   - week  
     -- name ..... FS2-220  
     -- number of month ..... FS2-239  
     -- number of year ..... FS2-236

-- setting.....	FS2-238
-- weekday .....	FS2-201
- year .....	
-- first day of .....	FS2-200
-- last day of .....	FS2-210
-- leap.....	FS2-211
Date functions .....	FS2-196
DeleteFile( ) .....	FS2-314
DirChange( ) .....	FS2-316
Directory .....	
- change .....	FS2-316
- create .....	FS2-318
- name of .....	FS2-320
- remove .....	FS2-322
DirMake( ) .....	FS2-318
DirName( ) .....	FS2-320
DirRemove( ) .....	FS2-322
DiskFree( ) .....	FS2-323
DiskTotal( ) .....	FS2-324
Dmy( ) .....	FS2-205
DosParam( ) .....	FS2-363
DoY( ) .....	FS2-207
DsetKbios( ) .....	FS2-283
DsetNoline( ) .....	FS2-283
DsetQfile( ) .....	FS2-283
DsetType( ) .....	FS2-292
DsetWindeb( ) .....	FS2-283
DsetWindow( ) .....	FS2-283
DtoR( ) .....	FS2-147

## **E**

Ega43( ) .....	FS2-283
EgaPalette( ) .....	FS2-283
Enhanced( ) .....	FS2-293
EnvParam( ) .....	FS2-363
EoM( ) .....	FS2-208
EoQ( ) .....	FS2-209
EoY( ) .....	FS2-210
ErrorAct( ) .....	FS2-363
ErrorBase( ) .....	FS2-363
ErrorCode( ) .....	FS2-363
ErrorOrg( ) .....	FS2-363
ExeName( ) .....	FS2-363
Expand( ) .....	FS2-68
Exponent( ) .....	FS2-171

## **F**

Fact( ) .....	FS2-148
Fahrenheit( ) .....	FS2-172
File .....	
- absolute path .....	FS2-362
- access rights .....	FS2-337
- append data .....	FS2-325
- attribute .....	
-- default .....	FS2-353
-- set .....	FS2-351
- attributes .....	FS2-326
- checksum of .....	FS2-329
- copy .....	FS2-330
- date .....	FS2-331
-- set .....	FS2-354
- delete .....	FS2-314, 332
- group of .....	FS2-334
- move .....	FS2-335
- owner .....	FS2-336
- read .....	
-- to string .....	FS2-341
- rename .....	FS2-335, 346
- search for .....	FS2-339
- seek .....	
-- restore .....	FS2-347
-- save .....	FS2-348
- size .....	FS2-340
- symbolic link .....	FS2-343
- temporary .....	FS2-360
- time .....	FS2-344
-- set .....	FS2-354
- valid name .....	FS2-345
- write string to .....	FS2-356
File system .....	
- free space .....	FS2-323
- total space .....	FS2-324
FileAppend( ) .....	FS2-325
FileAttr( ) .....	FS2-326
FileCheck( ) .....	FS2-329
FileCopy( ) .....	FS2-330
FileDate( ) .....	FS2-331
FileDelete( ) .....	FS2-332
FileGroup( ) .....	FS2-334
FileMove( ) .....	FS2-335
FileOwner( ) .....	FS2-336
FileRights( ) .....	FS2-337
FileScreen( ) .....	FS2-296
FileSeek( ) .....	FS2-339

FilesFree( )	FS2-363
FileSize( )	FS2-340
FilesMax( )	FS2-363
FileStr( )	FS2-341
FileSymLnk( )	FS2-343
FileTime( )	FS2-344
FileValid( )	FS2-345
FirstCol( )	FS2-283
FirstRow( )	FS2-283
Floor( )	FS2-149
FontLoad( )	FS2-283
FontReset( )	FS2-283
FontRotate( )	FS2-283
FontSelect( )	FS2-283
FS2 Toolbox	FS2-7
FtoC( )	FS2-173
Function	
- FS2 Toolbox	FS2-7
Fv( )	FS2-150

---

## G

GetBoxgrow( )	FS2-283
GetClearAttr( )	FS2-294
GetClearByte( )	FS2-295
GetCountry( )	FS2-363
GetCursor( )	FS2-283
GetFldCol( )	FS2-365
GetFldRow( )	FS2-365
GetFldVar( )	FS2-365
GetFont( )	FS2-283
GetInput( )	FS2-365
GetKxlat( )	FS2-283
GetKxtab( )	FS2-283
GetLines( )	FS2-283
GetMode( )	FS2-283
GetPage( )	FS2-283
GetPbios( )	FS2-283
GetPrec( )	FS2-151
GetPxlat( )	FS2-283
GetScrmode( )	FS2-283
GetScrstr( )	FS2-283
GetSecret( )	FS2-365
GetTab( )	FS2-283
GetTic( )	FS2-363
GetVgapal( )	FS2-283

---

## H

Hard disk	
- free space	FS2-323
- total space	FS2-324
HexToStr( )	FS2-69

---

## I

InByte( )	FS2-363
Infinity( )	FS2-174
InputMode( )	FS2-283
Integer	
- convert	
-- to float	FS2-168
IntNeg( )	FS2-175
IntPos( )	FS2-176
InvertAttr( )	FS2-283
InvertWin( )	FS2-283
InWord( )	FS2-363
IsAnsi( )	FS2-363
IsAt( )	FS2-363
IsBit( )	FS2-177
IsCga( )	FS2-283
IsDebug( )	FS2-363
IsEga( )	FS2-283
IsHercules( )	FS2-283
IsLeap( )	FS2-211
IsMath( )	FS2-363
IsMcga( )	FS2-283
IsMono( )	FS2-283
IsPga( )	FS2-283
IsVga( )	FS2-283

---

## J

JustLeft( )	FS2-70
JustRight( )	FS2-71

---

## K

KbdDisable( )	FS2-363
KbdEmulate( )	FS2-363
KbdSpeed( )	FS2-363
KbdStat( )	FS2-363
KbdType( )	FS2-363
Keyboard	

- buffer size .....	FS2-292
- simulate input .....	FS2-297
- trapping .....	FS2-307
- trigger	
-- after time delay .....	FS2-212
-- at clock time .....	FS2-214
Keyread( ) .....	FS2-283
Keysec( ) .....	FS2-363
KeySec( ) .....	FS2-212
KeySend( ) .....	FS2-297
Keytime( ) .....	FS2-363
KeyTime( ) .....	FS2-214
KsetCaps( ) .....	FS2-363
KsetIns( ) .....	FS2-363
KsetNum( ) .....	FS2-363
KsetScroll( ) .....	FS2-271, 363

## **L**

LastDayOm( ) .....	FS2-217
LastKfunc( ) .....	FS2-363
LastKline( ) .....	FS2-363
LastKproc( ) .....	FS2-363
Levenshtein Distance .....	FS2-107
Like( ) .....	FS2-72
Log10( ) .....	FS2-152
Logical	
- complement .....	FS2-58
- convert	
-- to numeric .....	FS2-178
LtoC( ) .....	FS2-73
LtoN( ) .....	FS2-178

## **M**

Mantissa( ) .....	FS2-179
Mathematical functions .....	FS2-137
MaxFont( ) .....	FS2-283
MaxLine( ) .....	FS2-74
MaxPage( ) .....	FS2-283
Mdy( ) .....	FS2-218
MemSize( ) .....	FS2-363
Millisec( ) .....	FS2-363
MoniSwitch( ) .....	FS2-283
Monochrome( ) .....	FS2-283

## **N**

NtoC( ) .....	FS2-180
NtoCdw( ) .....	FS2-220
NtoCmonth( ) .....	FS2-221
NtoColor( ) .....	FS2-300
Nul( ) .....	FS2-363
NumAnd( ) .....	FS2-181
NumAt( ) .....	FS2-76
Number	
- bits to char .....	FS2-164
- complement .....	FS2-58
- convert	
-- base .....	FS2-169
NumBuffers( ) .....	FS2-363
NumCol( ) .....	FS2-299
NumCount( ) .....	FS2-182
Numeric	
- angle from sine, cosine .....	FS2-143
- binary AND .....	FS2-181
- binary NOT .....	FS2-186
- binary OR .....	FS2-187
- binary XOR .....	FS2-190
- bit	
-- set .....	FS2-195
-- test .....	FS2-177
- cash present value .....	FS2-156
- Celsius to Fahrenheit .....	FS2-172
- clear bits .....	FS2-166
- convert	
-- degree to radians .....	FS2-147
-- from logical .....	FS2-178
-- radians to degrees .....	FS2-158
-- to other base .....	FS2-180
- cosine .....	FS2-145
- cosine arc .....	FS2-140
- cotangent .....	FS2-146
- counter .....	FS2-182
- effective interest rate .....	FS2-157
- exponent of float .....	FS2-171
- factorial .....	FS2-148
- Fahrenheit to Celsius .....	FS2-165
- float to string .....	FS2-173
- future value of capital .....	FS2-150
- high order byte .....	FS2-183
- integer	
-- signed/unsigned .....	FS2-176
-- unsigned/signed .....	FS2-175
- largest number possible .....	FS2-174

- logarithm .....	FS2-152
- low order byte.....	FS2-184
- mantissa of a float.....	FS2-179
- mirror 8 or 16bit value .....	FS2-185
- number of payment periods ...	FS2-154
- periodic payment amount.....	FS2-153
- pi value.....	FS2-155
- precision.....	FS2-151, 159
- pseudo-random numbers.....	FS2-191
- random numbers.....	FS2-193
- rotate 16-bit.....	FS2-188
- round	
-- down.....	FS2-149
-- up .....	FS2-144
- sign.....	FS2-160
- sine.....	FS2-161
- sine arc.....	FS2-141
- tangent .....	FS2-162
- tangent arc .....	FS2-142
NumFiles( ) .....	FS2-363
NumFkey( ) .....	FS2-363
NumHigh( ).....	FS2-183
NumLine( ) .....	FS2-77
NumLow( ) .....	FS2-184
NumMirr( ).....	FS2-185
NumNot( ) .....	FS2-186
NumOr( ) .....	FS2-187
NumRol( ).....	FS2-188
NumToken( ) .....	FS2-78
NumXor( ) .....	FS2-190

---

## O

OsVer( ) .....	FS2-363
OutByte( ) .....	FS2-363
OutWord( ) .....	FS2-363

---

## P

PadLeft( ) .....	FS2-79
PadRight( ).....	FS2-80
PageCopy( ) .....	FS2-283
Path	
- absolute.....	FS2-362
Payment( ) .....	FS2-153
PcType( ) .....	FS2-363
PeekByte( ) .....	FS2-363
PeekStr( ) .....	FS2-363

PeekWord( ).....	FS2-363
Periods( ) .....	FS2-154
Pi( ) .....	FS2-155
PokeByte( ) .....	FS2-363
PokeWord( ).....	FS2-363
PosAlpha( ) .....	FS2-81
PosChar( ).....	FS2-82
PosDel( ) .....	FS2-83
PosDiff( ) .....	FS2-84
PosEqual( ) .....	FS2-85
PosIns( ).....	FS2-86
PosLower( ).....	FS2-87
PosRange( ).....	FS2-88
PosRepl( ) .....	FS2-90
PosUpper( ).....	FS2-91
PrintError( ) .....	FS2-283
Pv( ) .....	FS2-156

---

## Q

Quarter( ) .....	FS2-222
------------------	---------

---

## R

Rand( ) .....	FS2-191
Random( ) .....	FS2-193
RangeRem( ) .....	FS2-92
RangeRepl( ) .....	FS2-94
Rate( ) .....	FS2-157
RemAll( ) .....	FS2-96
RemLeft( ) .....	FS2-97
RemRight( ).....	FS2-98
RenameFile( ) .....	FS2-346
ReplAll( ) .....	FS2-99
ReplLeft( ) .....	FS2-101
ReplRight( ).....	FS2-102
RestCursor( ) .....	FS2-283
RestFseek( ) .....	FS2-347
RestGets( ).....	FS2-365
RestSetkey( ) .....	FS2-365
RestToken( ) .....	FS2-103
RtoD( ) .....	FS2-158

---

## S

SaveCursor( ).....	FS2-283
SaveFseek( ).....	FS2-348



SaveGets( ) .....	FS2-365	SetPage( ) .....	FS2-283
SaveSetkey( ) .....	FS2-365	SetPbios( ) .....	FS2-283
SaveToken( ) .....	FS2-104	SetPrec( ) .....	FS2-159
SayDown( ) .....	FS2-283	SetPxlat( ) .....	FS2-283
SayMovein( ) .....	FS2-283	SetQname( ) .....	FS2-283
SayScreen( ) .....	FS2-283	SetRc( ) .....	FS2-283
SaySpread( ) .....	FS2-283	SetScrmode( ) .....	FS2-283
ScanKey( ) .....	FS2-363	SetScrstr( ) .....	FS2-283
Screen		SetTab( ) .....	FS2-283
- clear .....	FS2-290	SetTic( ) .....	FS2-363
-- by specif. char .....	FS2-287	SetTime( ) .....	FS2-226
-- char .....	FS2-305	Showkey( ) .....	FS2-363
-- color .....	FS2-294, 304	ShowTime( ) .....	FS2-227
-- to EOL .....	FS2-289	Sign( ) .....	FS2-160
--- by specif. char .....	FS2-285	Sin( ) .....	FS2-161
- clear byte .....	FS2-294, 295	Sound( ) .....	FS2-363
- color		Source	
-- standard .....	FS2-306	- fs2_com.prg .....	FS2-311
-- unselected .....	FS2-309	- fs2_udfs.prg .....	FS2-283
- column .....	FS2-299	Speed( ) .....	FS2-363
- restore from file .....	FS2-296	SsetBreak( ) .....	FS2-363
- save to file .....	FS2-301	SsetVerify( ) .....	FS2-363
- size of saved file .....	FS2-303	StackFree( ) .....	FS2-363
ScreenAttr( ) .....	FS2-283	Standard( ) .....	FS2-306
ScreenFile( ) .....	FS2-301	StrDiff( ) .....	FS2-107
ScreenMark( ) .....	FS2-283	StrFile( ) .....	FS2-356
ScreenMix( ) .....	FS2-283	String	
ScreenSize( ) .....	FS2-303	- 2byte sequence to 1byte .....	FS2-136
ScreenStr( ) .....	FS2-283	- add value to all chars .....	FS2-29
SecToTime( ) .....	FS2-223	- adds value to char .....	FS2-11
Serial		- adjust beginning position .....	FS2-16
- communication .....	FS2-310	- ASCII value at specif. posit. ....	FS2-15
SetAtlike( ) .....	FS2-105	- before n-th sequence .....	FS2-24
SetBell( ) .....	FS2-283	- binary AND to all chars .....	FS2-30
SetBit( ) .....	FS2-195	- binary NOT to all chars .....	FS2-36
SetBoxgrow( ) .....	FS2-283	- binary OR to all chars .....	FS2-40
SetClearAttr( ) .....	FS2-304	- binary XOR to all chars .....	FS2-54
SetClearByte( ) .....	FS2-305	- bits to char .....	FS2-164
SetDate( ) .....	FS2-225	- bytes to hex value .....	FS2-111
SetFattr( ) .....	FS2-351	- center .....	FS2-27
SetFcreate( ) .....	FS2-353	- checksum .....	FS2-56
SetFdati( ) .....	FS2-354	- common denominator .....	FS2-39
SetFont( ) .....	FS2-283	- compare chars .....	FS2-72
SetKxlat( ) .....	FS2-283	- complement .....	FS2-58
SetKxtab( ) .....	FS2-283	- compress .....	FS2-42
SetLastKey( ) .....	FS2-363	- convert	
SetLines( ) .....	FS2-283	-- base .....	FS2-169
SetMaxcol( ) .....	FS2-283	-- from date .....	FS2-205, 218
SetMaxrow( ) .....	FS2-283	-- to bit pattern .....	FS2-167

- to numeric ..... FS2-168
- convert from logical ..... FS2-73
- correlated replacement ..... FS2-45
- correlation ..... FS2-44
- count particular chars left ..... FS2-59
- count particular chars right ..... FS2-60
- CRC 16 value ..... FS2-61
- crypt with password ..... FS2-63
- delete at particular position ..... FS2-83
- delete chars of specif. range .... FS2-92
- delete multiple 2byte sequence ... FS2-131
- every even position ..... FS2-31
- every odd position ..... FS2-37
- expand at separator ..... FS2-51
- expand tabs ..... FS2-112
- find denominator ..... FS2-132
- find difference ..... FS2-84
- find equivalence ..... FS2-85
- find lower case char ..... FS2-87, 88
- find numerical value ..... FS2-130
- find token ..... FS2-22
- find upper case char ..... FS2-91
- hex value into bytes ..... FS2-69
- insert at particular position ..... FS2-86
- insert chars ..... FS2-68
- Levenshtein Distance ..... FS2-107
- list of characters not used ..... FS2-35
- list of characters used ..... FS2-32
- longest line in text ..... FS2-74
- mirrors characters ..... FS2-33
- mix two strings ..... FS2-34
- move left chars to end ..... FS2-70
- move right chars to begin ..... FS2-71
- multi-pass At() ..... FS2-65
- number of lines in text ..... FS2-77
- number of occurrences ..... FS2-76
- number of tokens ..... FS2-78
- position of first alpha ..... FS2-81
- remove chars ..... FS2-96, 97, 98
- remove duplicate chars ..... FS2-38
- remove particular chars ..... FS2-46
- replace 2byte sequence ..... FS2-133
- replace at particular position .... FS2-82
- replace chars ..... FS2-90, 99, 101, 102
- replace chars of specif. range .. FS2-94
- replace particular chars ..... FS2-47
- rest of found occurence ..... FS2-12
- search & ..... FS2-see replace

- search for n-th substring ..... FS2-18
- setting for At() ..... FS2-105
- sort sequence ..... FS2-49
- space to tabs ..... FS2-114
- sum of string ASCII values ..... FS2-14
- swap 2byte sequence ..... FS2-135
- swap chars ..... FS2-52
- swap two ..... FS2-109
- tabs to space ..... FS2-112
- token
  - array of ..... FS2-118
  - availability ..... FS2-121
  - initialize ..... FS2-122
  - lower case ..... FS2-124
  - next ..... FS2-125
  - recent ..... FS2-120
  - restore ..... FS2-103
  - save ..... FS2-104
  - select ..... FS2-116
  - separator ..... FS2-127, 128
  - upper case ..... FS2-129
- uncompress ..... FS2-53
- StrScreen( ) ..... FS2-283
- StrSwap( ) ..... FS2-109
- StrToHex( ) ..... FS2-111

---

## T

- TabExpand( ) ..... FS2-112
- TabPack( ) ..... FS2-114
- Tan( ) ..... FS2-162
- TempFile( ) ..... FS2-360
- Time
  - check if expired ..... FS2-235
  - convert
    - to seconds ..... FS2-229
  - display continuously ..... FS2-227
  - seconds
    - from time ..... FS2-229
  - seconds into a time ..... FS2-223
  - set ..... FS2-226
  - trigger
    - UDF ..... FS2-233
  - validation ..... FS2-231
- Time functions ..... FS2-196
- TimeToSec( ) ..... FS2-229
- TimeValid( ) ..... FS2-231
- Token( ) ..... FS2-116

TokenArray( )	FS2-118
TokenAt( )	FS2-120
TokenEnd( )	FS2-121
TokenInit( )	FS2-122
TokenLower( )	FS2-124
TokenNext( )	FS2-125
TokenSep( )	FS2-127
TokenSepDef( )	FS2-128
TokenUpper( )	FS2-129
ToolVer( )	FS2-363
TrapAnyKey( )	FS2-307
TrapInput( )	FS2-283
TrapShift( )	FS2-283
Trigger	
- display time	FS2-227
- UDF	
-- at time	FS2-233
TriggerUdf( )	FS2-233
TrueName( )	FS2-362

---

## U

Unselected( )	FS2-309
UnsetTextWin( )	FS2-283

---

## V

ValPos( )	FS2-130
Vga28( )	FS2-283
Vga50( )	FS2-283
VgaPalette( )	FS2-283
Video	
- introduction	FS2-282
VideoInit( )	FS2-283
VideoSetup( )	FS2-283
VideoType( )	FS2-283

---

## W

WaClose( )	FS2-243
WaitPeriod( )	FS2-235
Wboard( )	FS2-244
Wbox( )	FS2-245
Wcaption( )	FS2-247
Wcenter( )	FS2-248
Wclose( )	FS2-249
Wcol( )	FS2-250

Week( )	FS2-236
WeekSetIso( )	FS2-238
WfCol( )	FS2-252
WfLastCol( )	FS2-253
WfLastRow( )	FS2-254
Wformat( )	FS2-255
WfRow( )	FS2-257

## Window

- area usable	FS2-255
- border mode	FS2-265
- caption in frame	FS2-247
- center	FS2-248, 264
- close	FS2-249
- close all	FS2-243
- column	FS2-258
-- formatted	FS2-252, 253
-- frame	FS2-250
-- max	FS2-260
- frame box	FS2-245
- ID last	FS2-273
- move	FS2-266, 267, 268, 269, 270
-- on/off	FS2-271, 279
-- steps	FS2-281
- open	FS2-274
- row	FS2-259
-- formatted	FS2-254, 257
-- frame	FS2-277
-- max	FS2-262
- screen area	FS2-244
- select	FS2-278
- shadow color	FS2-280

## Windowing

- introduction	FS2-240
WlastCol( )	FS2-258
WlastRow( )	FS2-259
WmaxCol( )	FS2-260, 298
WmaxRow( )	FS2-262, 298
Wmiddle( )	FS2-264
Wmode( )	FS2-265
Wmove( )	FS2-266
WmoveDown( )	FS2-267, 270
WmoveLeft( )	FS2-268
WmoveRight( )	FS2-269
WmoveUser( )	FS2-271
Wnum( )	FS2-273
WoM( )	FS2-239
Wopen( )	FS2-274
WordOne( )	FS2-131
WordOnly( )	FS2-132

WordRepl( ).....FS2-133  
WordSwap( ).....FS2-135  
WordToChar( ).....FS2-136  
Wrow( ) .....FS2-277  
Wselect( ).....FS2-278  
WsetMove( ).....FS2-279  
WsetShadow( ) .....FS2-280  
Wstep( ) .....FS2-281

---

**x**

XMOBLOCK( ) .....FS2-310  
XMOCHECK( ).....FS2-310  
Xtoc( ) .....FS2-363

---

**z**

ZEROINSERT( ) .....FS2-310  
ZEROREMOVE( ).....FS2-310





---

**multisoft Datentechnik**  
**Harthausen Str. 85**  
**D-81545 München**

**<http://www.fship.com>**  
**[sales@multisoft.de](mailto:sales@multisoft.de)**  
**[support@flagship.de](mailto:support@flagship.de)**