

DynaComm  
Connectivity  
Series®

---

*Script Language Reference*

DCS

© 2012 by FutureSoft, Inc. All rights reserved.

## DynaComm Connectivity Series™ Script Language Reference

This manual, and the software described in it, is furnished under a license agreement. Information in this document is subject to change without notice and does not represent a commitment on the part of FutureSoft. The software may be used or copied only in accordance with the terms of the agreement.

No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose other than the purchaser's personal use without the written permission of FutureSoft, Inc.

DynaComm, DynaComm Connectivity Series, DynaComm Client Option, and FutureSoft are registered trademarks of FutureSoft, Inc. All other trademarks are the property of their respective owners.

1986-2012 FutureSoft, Inc.

E-Edition 14-2012  
Document #E-DCSR9200 013009

Written and designed at:  
FutureSoft, Inc.

(800) 989-8908

[info@futuresoft.com](mailto:info@futuresoft.com)

<http://www.futuresoft.com>

---

---

# Contents

---

## 1

### Introduction to the Script Language

.....	15
DCS Script Language Overview.....	16
Creating, Compiling, and Executing a Script .....	17
Commands, Functions and Arguments.....	20
Command Blocks .....	22
Line Continuation.....	23
Comments.....	24
Labels .....	25
Strings.....	27
Numerics.....	37
Booleans.....	40
Special Argument Types.....	43
Variable Creation.....	47
Symbols.....	49
Operators.....	54
Scoping Rules .....	59
Parameter Passing & Subroutines .....	61
Tables, Records and Data Manipulation .....	64
Menus .....	71
DCS Windows & Window Handles .....	74
Event Handling - WAIT and WHEN Commands.....	76
Dynamic Data Exchange - DDE.....	78
Task Errors.....	82
Converting Scripts from Previous Versions of DCS.....	84

## 2

### Functions

.....	87
Functions in Alphabetical Order .....	88
Functions by Category.....	92
ACTIVE.....	96
ATTRIBUTES.....	97
BAND .....	98
BNOT.....	99
BOOL.....	100

---

## Contents, *continued*

---

### 2

#### Functions

Functions, *continued*

BOR.....	101
BUFFER.....	102
BUFFER, continued .....	103
BXOR.....	104
CHR.....	105
CONNECT .....	107
CONNECTMESSAGE.....	108
CONNECTRESULT .....	109
CURSOR .....	110
DATE.....	111
(DDE) ADVISE.....	112
DECRYPT.....	113
DEFAULTSESSIONHANDLE .....	115
(DIALOG) CHECKBOX .....	116
(DIALOG) EDITTEXT .....	117
(DIALOG) LISTBOX.....	118
(DIALOG) MESSAGEBOX.....	119
(DIALOG) RADIOGROUP .....	121
DIALOGHANDLE .....	122
DIRECTORY .....	123
DISKSPACE .....	125
ENCRYPT.....	126
EOF .....	127
ERROR.....	128
EXFLDATTR .....	129
EXISTS.....	134
FILESIZE .....	135
FILTER .....	136
FLDATTR .....	138
FLDATTREXPOS.....	140
FLDLEN .....	141
FLDNUM .....	142
FLDPOS.....	143
FLDTEXT .....	144
GETAPPCONFIG.....	145
GETCONNCONFIG .....	146
GETDISPLAYCONFIG.....	147

---

## Contents, *continued*

---

### 2

#### Functions

Functions, *continued*

GETEMULCONFIG.....	148
GETGENERALCONFIG.....	149
GETPROFILEDATA.....	150
GETXFERCONFIG.....	151
HWNDLIST.....	153
ICONIC.....	155
INT.....	156
LENGTH.....	157
(MENU) CHECKED.....	158
(MENU) ENABLED.....	159
NETID.....	160
NEXT.....	161
NUM.....	162
ORD.....	163
PASSWORD.....	164
PHONENUMBER.....	165
POS.....	166
POSITION.....	167
POWER.....	168
PRTMETRICS.....	169
PUTPROFILEDATA.....	170
RANDOM.....	172
REAL.....	173
RESULT.....	174
ROUND.....	175
ROUTE.....	176
SCREEN.....	178
SEARCH.....	180
SEARCHINRECT.....	182
SECONDS.....	183
SETTINGS.....	184
STR.....	192
SUBSTR.....	193
SYSMETRICS.....	194
SYSTEM.....	195
TASKFILE.....	197
TASKLIST.....	198

---

## Contents, *continued*

---

### 2

#### Functions

Functions, *continued*

TASKNAME.....	199
TIME .....	200
TIMER.....	201
TRIM .....	202
TYPEDLIBRARYCALL.....	203
UPPER.....	205
USERID .....	206
VERSION .....	207
VISIBLE.....	208
WINDOW .....	209
WINDOWHND.....	210
WINDOWNAME.....	211
WNDCLASS.....	212
WNDFILE.....	213
WNDTITLE .....	214
ZOOMED .....	215

### 3

#### Commands

.....	216
Commands in Alphabetical Order .....	217
Commands by Category.....	227
APPCONFIG .....	240
ARGUMENTS .....	242
BEEP .....	243
BEGIN.....	244
BREAK.....	245
CANCEL.....	246
CLEAR.....	247
COLLECT .....	248
COLLECT, continued .....	249
COMPILE.....	251
CONCAT .....	252
CONNCONFIG.....	253
CONNECT .....	256

---

## Contents, *continued*

---

### 3

#### Commands

Commands, *continued*

CONTINUE.....	258
CREATE DIRECTORY .....	259
(DDE) ACCESS .....	260
(DDE) ACCESS CANCEL.....	262
(DDE) INSTRUCT .....	263
(DDE) POKE.....	264
(DDE) REQUEST .....	265
(DDE) TABLE REPLY .....	266
(DDE) TABLE REQUEST .....	268
(DDE) TABLE SEND .....	269
(DDE) WAIT SIGNAL .....	270
(DDE) WHEN ADVISE .....	271
(DDE) WHEN EXECUTE.....	273
(DDE) WHEN INITIATE.....	275
(DDE) WHEN POKE .....	276
(DDE) WHEN REQUEST .....	278
(DDE) WHEN TERMINATE .....	280
DEBUG.....	281
DECREMENT .....	283
DIAL.....	284
DIALOG .....	286
(DIALOG) BUTTON .....	291
DIALOG CANCEL.....	293
(DIALOG) CHECKBOX .....	294
DIALOG CONTROL .....	296
(DIALOG) DIMENSION.....	300
(DIALOG) EDITTEXT .....	301
(DIALOG) GROUPBOX .....	303
(DIALOG) ICON .....	304
(DIALOG) ICONBUTTON .....	305
(DIALOG) LISTBOX.....	307
(DIALOG) MESSAGE .....	309
(DIALOG) NEWLINE.....	310
(DIALOG) PICTURE .....	311
(DIALOG) RADIOBUTTON .....	312
(DIALOG) RADIOGROUP .....	313
DIALOG UPDATE .....	315

---

## Contents, *continued*

---

### 3

#### Commands

Commands, *continued*

DISCONNECT .....	318
DISPLAY .....	319
DISPLAYCONFIG.....	321
DROPDTR .....	323
EDIT COPY .....	324
EDIT COPYSPECIAL .....	325
EDIT CUT.....	328
EDIT FIND .....	329
EDIT GOTO .....	330
EDIT PASTE.....	331
EDIT REPLACE .....	332
EMULCONFIG.....	334
END .....	343
EXECUTE .....	344
FILE CLOSE .....	345
FILE COMPRESS.....	346
FILE COPY .....	347
FILE CREATENAME .....	348
FILE DECOMPRESS .....	351
FILE DECRYPT .....	352
FILE DELETE.....	353
FILE ENCRYPT.....	354
FILE OPENNAME .....	355
FILE PAUSE .....	357
FILE RECEIVE BINARY.....	358
FILE RENAME.....	361
FILE RESUME .....	362
FILE SEND BINARY .....	363
FKEYS.....	366
GENERALCONFIG.....	367
GOTO .....	369
HANGUP .....	370
IF.....	371
INCREMENT .....	372
KERMIT COPY .....	373
KERMIT DIRECTORY .....	374
KERMIT ERASE .....	375



---

## Contents, *continued*

---

### 3

#### Commands

Commands, *continued*

KERMIT FINISH.....	376
KERMIT FREESPACE.....	377
KERMIT HELP.....	378
KERMIT LOGOUT.....	379
KERMIT MESSAGE.....	380
KERMIT NEWDIRECTORY.....	381
KERMIT RENAME.....	382
KERMIT TYPE.....	383
KERMIT WHO.....	384
KEY.....	385
KEYBOARD.....	389
KEYMAP LOAD.....	390
KEYMAP RESET.....	391
KEYMAP SAVE.....	392
LAUNCH.....	393
LEAVE.....	395
LEVEL.....	396
LIBRARY CALL.....	397
LIBRARY LOAD.....	400
LIBRARY UNLOAD.....	401
LINENUMBERS.....	402
LOAD.....	403
LOGTOFILE.....	404
MENU.....	405
MENU CANCEL.....	406
MENU DELETE ITEM.....	407
MENU DELETE POPUP.....	408
MENU INSERT ITEM.....	409
MENU INSERT POPUP.....	411
(MENU) ITEM.....	412
(MENU) POPUP.....	414
(MENU) SEPARATOR.....	416
MENU UPDATE.....	417
NOSHOW.....	419
PARSE.....	420
PERFORM.....	422
PRINT CANCEL.....	423

---

## Contents, *continued*

---

### 3

#### Commands

Commands, *continued*

PRINT CLOSE .....	424
PRINT FILE .....	425
PRINT FONT .....	426
PRINT NEWLINE.....	427
PRINT NEWPAGE .....	428
PRINT OPEN.....	429
PRINT STRING .....	430
PRINT STYLE .....	431
PRINT TABS .....	432
PRINT TERMINAL.....	433
QUIT .....	434
RECORD FORMAT .....	435
RECORD READ .....	436
RECORD SCAN .....	438
RECORD WRITE.....	439
REMOVE DIRECTORY .....	441
RESETSERIAL .....	442
RESTART .....	443
RESUME .....	444
RETURN.....	445
SAVE .....	446
SCREEN .....	447
SCROLL DOWN.....	448
SCROLL LEFT .....	449
SCROLL RIGHT.....	450
SCROLL UP.....	451
SELECTION.....	452
SELECTION APPEND.....	453
SELECTION BUFFER .....	454
SELECTION PRINT .....	455
SELECTION SAVE.....	456
SELECTION SEND.....	457
SEND .....	458
SEENDBREAK.....	463
SET .....	464
SET APPTITLE .....	465
SET ATTRIBUTES.....	466

---

## Contents, *continued*

---

### 3

#### Commands

Commands, *continued*

SET AUTOSCROLLTOCURSOR.....	467
SET AUTOSIZE.....	468
SET BACKSPACEDESTRUCTIVE.....	469
SET BACKSPACEKEY.....	470
SET BAUDRATE.....	471
SET BINARYTRANSFERPARAMS.....	472
SET BINARYTRANSFERS.....	474
SET BUFFERLINES.....	475
SET CARRIERDETECT.....	476
SET COLUMNS.....	477
SET CONNECTION.....	478
SET CONNECTMESSAGE.....	480
SET CONNECTRESULT.....	481
SET CURSOR.....	482
SET DATABITS.....	483
SET DDETIMEOUT.....	484
SET DECIMAL.....	485
SET DEFAULTSESSIONHANDLE.....	486
SET DIRECTORY.....	487
SET EMULATION.....	488
SET FKEYSSHOW.....	490
SET FLOWCONTROL.....	491
SET KEEPPRINTCHANNELOPEN.....	492
SET LOCALECHO.....	494
SET NETID.....	495
SET OUTGOINGCR.....	496
SET PARITY.....	497
SET PASSTHROUGH.....	498
SET PASSWORD.....	499
SET PHONENUMBER.....	500
SET RESULT.....	501
SET RETRY.....	502
SET RETRYDELAY.....	503
SET SENDDELAY.....	504
SET SIGNAL.....	505
SET SOUND.....	506
SET STOPBITS.....	507

---

## Contents, *continued*

---

### 3

#### Commands

Commands, *continued*

SET TERMCLOSE.....	508
SET TERMFONT.....	509
SET USERID.....	510
SET WILDCARD.....	511
SET WINDOWTITLE.....	512
SET WORDWRAP.....	513
SET XCLOCK.....	514
SET XSYSTEM.....	515
SETTINGS.....	516
SHOW.....	517
SPAWN.....	518
SWITCH.....	519
SYSTEM.....	521
TABLE CLEAR.....	523
TABLE CLOSE.....	524
TABLE COPY.....	525
TABLE DEFINE.....	527
TABLE LOAD.....	529
TABLE SAVE.....	530
TABLE SORT.....	531
TASKERROR.....	532
TASKSTOP.....	534
TIMER RESET.....	535
TITLE.....	536
TOOLBARHIDE.....	537
TOOLBARSHOW.....	538
TRANSFERS.....	539
WAIT CHAR.....	544
WAIT CLOSE.....	545
WAIT DELAY.....	546
WAIT ECHO.....	548
WAIT EDIT.....	549
WAIT PROMPT.....	550
WAIT QUIET.....	551
WAIT RESUME.....	553
WAIT SCREEN.....	554
WAIT STRING.....	555

---

## Contents, *continued*

---

### 3

#### Commands

*Commands, continued*

WAIT UNTIL.....	556
WHEN CANCEL.....	557
WHEN COLLECT.....	559
WHEN DISCONNECT.....	561
WHEN ECHO.....	562
WHEN ERROR.....	563
WHEN INPUT.....	564
WHEN QUIET.....	566
WHEN SCREEN.....	567
WHEN STRING.....	569
WHEN TIMER.....	571
WHEN WINDOW.....	572
WHILE.....	573
WINDOW ACTIVATE.....	574
WINDOW ARRANGE.....	575
WINDOW CLOSE.....	576
WINDOW DEFAULT.....	577
WINDOW HIDE.....	578
WINDOW MAXIMIZE.....	579
WINDOW MESSAGE.....	580
WINDOW MINIMIZE.....	581
WINDOW MOVE.....	582
WINDOW OPEN.....	583
WINDOW RESTORE.....	585
WINDOW STACK.....	586
WINDOW UNHIDE.....	587
XFERCONFIG.....	588

#### Appendices

A Task Errors.....	593
.....	593
Task Errors.....	594
Task Errors, continued.....	595
Task Errors, continued.....	596
Task Errors, continued.....	597

---

---

Task Errors, continued .....	598
<b>B New and Removed Commands and Functions .....</b>	<b>599</b>
.....	599
New Commands and Functions .....	600
Removed Commands and Functions .....	601
<b>C Quick Reference for Command and Function Syntax .....</b>	<b>603</b>
.....	603
<b>Index.....</b>	<b>619</b>

# 1

---

## *Introduction to the Script Language*

DCS

---

## DCS Script Language Overview

---

The DCS Script Language is one of most powerful features of the DCS Connectivity Series. It provides for the automation of frequently performed tasks while allowing for customizing DCS to the needs of your organization. For example, you can:

- ▼ Present data from a host machine in a Windows interface that you design.
- ▼ Send commands and data to a host machine.
- ▼ Transfer files.
- ▼ Print data from a host machine locally.
- ▼ Access other applications using Dynamic Data Exchange (DDE).
- ▼ Execute complex logic.
- ▼ Respond to queries from a host machine.
- ▼ Simplify complex interactions with the host machine.

### What You Will Learn

This chapter provides a general overview of DCS Script Language components beginning with script file management concepts. The overview includes:

- ▼ Script statement syntax
- ▼ A discussion of labels, comments, functions, and commands
- ▼ An explanation of arguments and variables
- ▼ Information about basic programming practices
- ▼ Script debugging and error correction

Each DCS function and command are discussed in detail in **Chapter 2 Functions** and **Chapter 3 Commands**, respectively.



---

## Creating, Compiling, and Executing a Script

---

The DCS Script Language is a compiled language. An operational script includes two files:

▼ Source file

The Source file is simply a text file containing all the script language commands necessary to accomplish the script purpose. It can easily be created using DCS editing capabilities, but can also be created using any type of program editor. It is a human-readable file, and cannot be executed. The Source file is given a “.dcp” extension.

▼ Task file

The Task file is created from the source file. It contains DCS internal machine language generated from the source file. It is not a human-readable file, and is the only type of file that DCS can execute. The Task file is given a “.dct” extension.

Scripts are created, edited, compiled, and executed in DCS with selections on the **File**, **Edit**, and **Script** menus.

### Creating and Editing Scripts

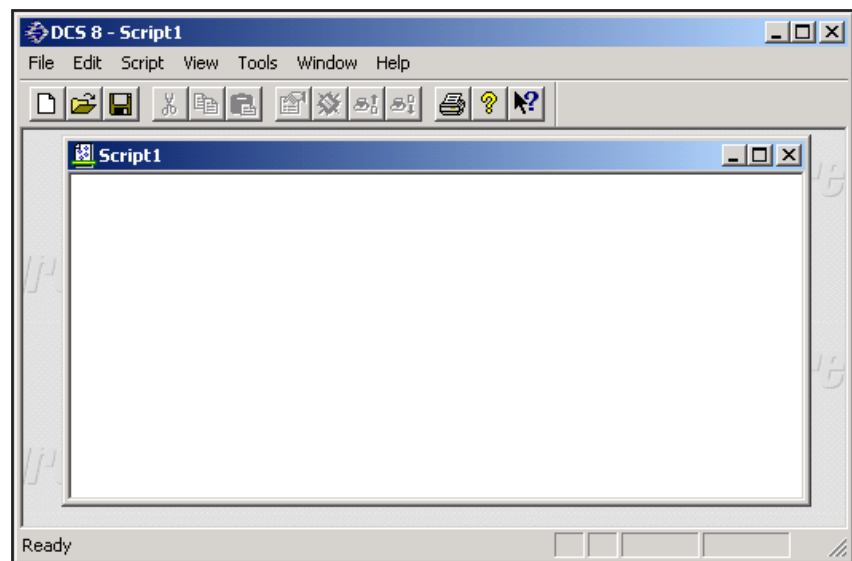
The first step in creating a new script is to create the Source document. Use the **New** selection on the **File** menu to create the script document.

▼ To create a new script

- 1 On the **File** menu, select **New**.
- 2 On the expanded **File** menu, select **Script**.

An empty script file opens in the DCS window. All commands are entered in this file.

Figure 1.1  
New script  
document



To create a script that automatically executes whenever DCS opens, save the script with the name “autostrt.dcp”. When DCS starts, a task file in the Futuresoft\DCSeries\scripts directory with the name “autostrt.dcp” is executed.

---

## Creating, Compiling, and Executing a Script, *continued*

---

To edit a script, use the options available on the **Edit** menu.

### Compiling and Executing

Before a script can be executed, it must be compiled. Compiling converts a script from a human-readable form (a source file) to a machine-readable form (a task file).

- ▼ To compile a script
  - 1 On the **Script** menu, select **Compile**.

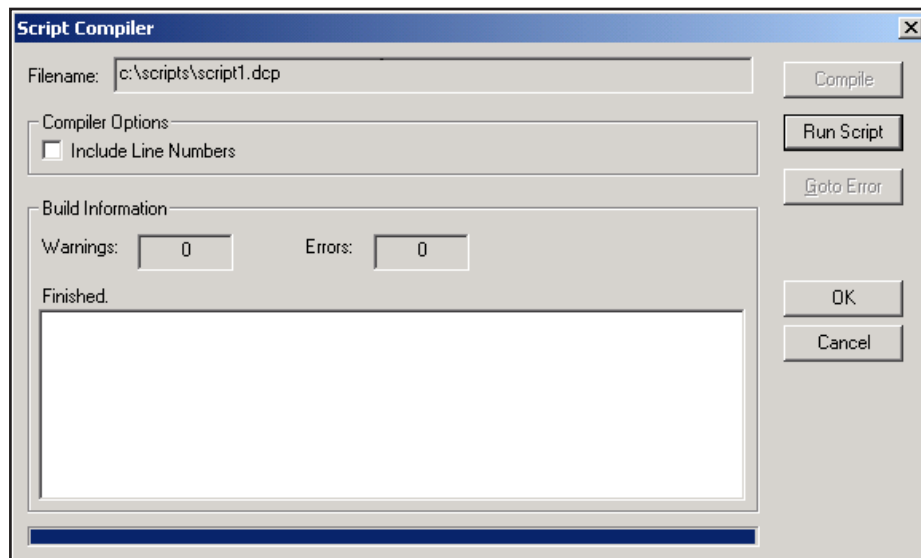
If:

- ▼ You are currently editing a script that has not been previously saved, you are prompted with the **Save As** dialog to save the script. Enter a name for the script and click **Save**. The **Script Compiler** dialog appears.
- ▼ You are not currently editing a script, or if the active window is not a script window, you are prompted with the **Open** dialog to open a previously saved script source (\*.dcp) file. Select the name of the file to compile and click **OK**. The **Script Compiler** dialog appears.
- ▼ The script you are editing has already been saved, the **Script Compiler** dialog immediately appears.

### Script Compiler Dialog

The **Script Compiler** dialog is used to compile a script document. The Script Compiler dialog appears in the DCS application window. It includes the name of the source file and a check box option for including line numbers.

Figure 1.2  
Script Compiler  
dialog



While DCS compiles the script, it checks for syntax errors. If no syntax errors are encountered, the script is successfully compiled and saved as a task file with the .dct extension. The **Script**

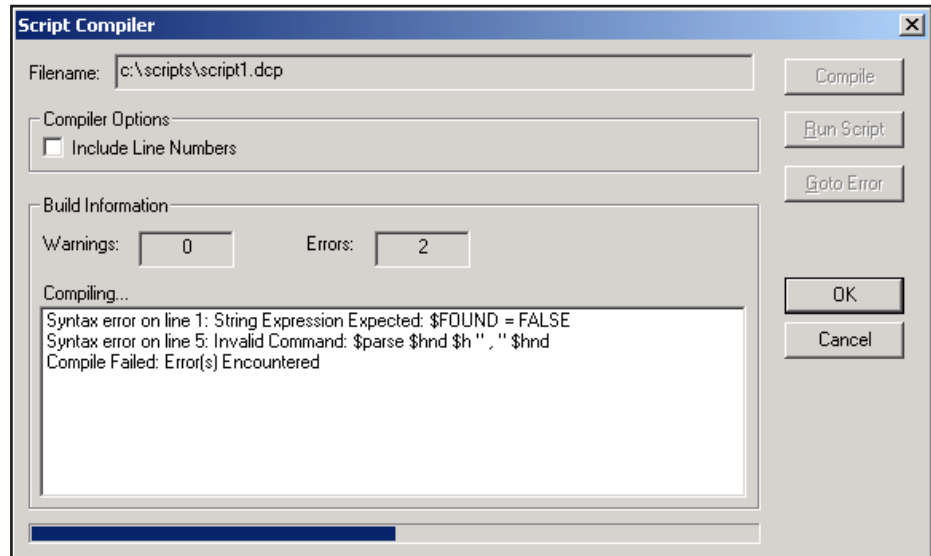
---

## Creating, Compiling, and Executing a Script, *continued*

---

**Compiler** dialog disappears. If DCS encounters a syntax error, a message identifying the type and location of the error appears in the **Script Compiler** dialog.


Figure 1.3  
Script Compiler  
dialog with er-  
rors listed



If you enable (check) **Include Line Numbers**, line numbers are placed in the compiled script. If errors are encountered during compilation, the line number of the offending command is provided to help find the command in the script.

 Also see: **SHOW** command

Highlight the error to correct and click **Go To Error** to jump to the error in the script. DCS shifts the focus to the script source and places the cursor at the location of the error. Click **Stop** to remove the **Script Compiler** dialog and return to the script window.

 **Note:** When a script is compiled, any changes made to the script are automatically saved. If you make major changes to a script, use **Save As** on the **File** menu before compiling the script and save the source file using a different name. If you do not, you will be unable to recover any previous versions of the script file.

When a script has been successfully compiled, the script can be executed at any time by selecting **Run** on the **Script** menu. The source script does not need to be available for the task file to execute. DCS executes the task file and stops if a task error is encountered or if **Stop** on the **Script** menu is selected.

---

## Commands, Functions and Arguments

---

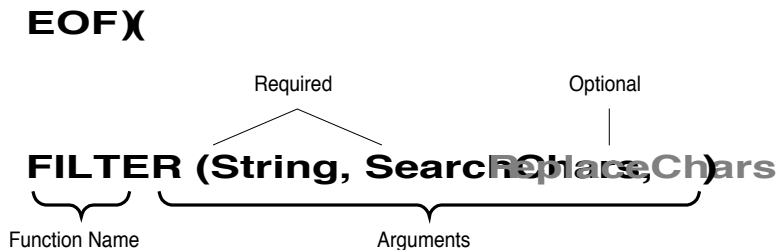
DCS Script Language statements are entered into a script document. Commands are executable statements that perform specific actions. Most commands accept *arguments* that specify how the actions are to be performed. Functions are executable statements that return a value, and therefore may be used as commands or as part of an assignment statement. Command and function arguments can be specified by either single operands, complex operands, or expressions of the specified type.

DCS Script Language statements require use of a specific syntax. Below are examples of a function statement and a command statement.

### Function Statement Syntax

Function statements begin with a function name and may include arguments. Function arguments must be enclosed in parentheses.

**Figure 1.4**  
Examples of  
two function  
statements

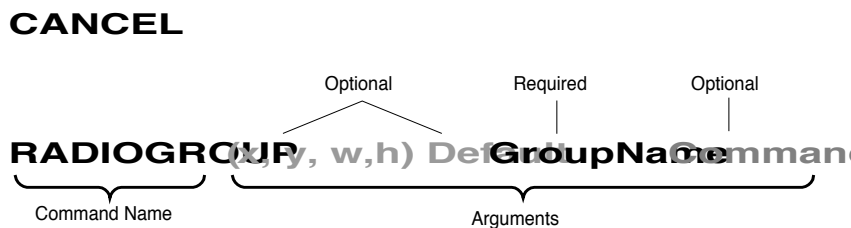


In Figure 1.4 above, the EOF function does not include any arguments. However, the () set must be included in the statement. Also in Figure 1.4, the FILTER statement illustrates that some arguments are required and some are optional.

### Command Statement Syntax

Command statements begin with a command name and typically include arguments. However, not all commands include arguments as seen in Figure 1.5 below with the CANCEL command.

**Figure 1.5**  
Examples of  
two command  
statements



Notice in Figure 1.5 that some command arguments are enclosed in parentheses while others are not. Notice too that some command arguments are required while others are optional.

---

## Commands, Functions and Arguments, *continued*

---

### Expressions

Expressions consist of a single or multiple operands combined with the appropriate operator for the specified type. Three types of operands are available:

- ▼ String
- ▼ Numeric
- ▼ Boolean

The assignment operator (equivalent to the **SET** command) assigns a value to a variable. Values can be numerics or strings (depending on the type of variable), or can be functions or expressions which return a value.

```
Variable = Value ; Example Syntax
```



Also see: **Variable Creation** in this chapter

### Example

This command:

```
%number = 2400
```

assigns the value “2400” (a single operand) to the integer variable %number.

This command:

```
$name = "Tim"
```

assigns the string value “Tim” to the string variable \$name.

This command:

```
%availSpace = DISKSPACE ()
```

assigns a value to the integer variable %availSpace. The **DISKSPACE** function can be used to specify the value argument since it returns a numeric value.

This command:

```
%val = (200/25) + (8*15)
```

assigns the value of the expression “(200/25) + 8\*15” to the variable %val.

---

## Command Blocks

---

Several methods are available to continue a script line to the next physical line if needed. The method used depends on the type of line to be continued.

### Command Blocks

A command block is a set of commands that is treated as a single logical command. Command blocks can be used to specify the `COMMAND` argument required by an `IF`, `ELSE`, `WHEN`, `WHILE`, or `DIALOG` control command.

Command blocks consist of either:

- ▼ a set of commands separated by commas, or
- ▼ a set of commands preceded by a `BEGIN` command and followed by an `END` command.

If the command block is used to specify a `WHEN` command argument and consists of commands separated by commas, the first command in the block must be on the same line as the `WHEN` command.

These examples show two ways of writing the same script fragment:

#### Example 1

```
WHEN QUIET "5" INCREMENT %timeout_cnt
DISPLAY (0,0) "time out: " | STR (%timeout_cnt)
IF (%timeout_cnt > 10)
    DISPLAY (1,0) "terminating process"
RESUME
```

#### Example 2

```
WHEN QUIET "5"
BEGIN
    INCREMENT %timeout_cnt
    DISPLAY (0,0) "time out: " | STR (%timeout_cnt)
    IF (%timeout_cnt > 10)
        BEGIN
            DISPLAY (1,0) "terminating process"
            RESUME
        END
    END
END
```

The command block is executed only when the `WHEN QUIET` command is activated.

---

## Line Continuation

---

Several methods are available for continuing lines within a script.

- ▼ To continue a line between script commands

To place more than one command on a line, use commas as shown below.

### Example

```
IF $date = "03/25/2001", $month = "March", $day = "25"
```



Caution! Extensive use of commas may make it difficult to locate specific statements and command blocks in the script.

- ▼ To continue a line between operands in an expression

Place the appropriate operator after the last operand and continue with the next operand on the following line. The valid operator for strings is the concatenation operator (`()`). The valid operators for numerics are `+`, `-`, `*`, `/`, and `%`. The valid operators for Boolean variables and expressions are `AND` and `OR`.

### Example

```
%sum = %apple + %orange + %peach + %banana
```

- ▼ To continue a line in the middle of a command

Place a backslash (`\`) at the end of the line and continue on the next line.

### Example

```
TABLE DEFINE 0 FIELDS CHAR 10 CHAR 20 INT 15 \  
INT 3 INT 15
```

- ▼ To continue a line containing `IF`, `ELSE` or `WHILE` commands

The lines of an `IF`, `ELSE` or `WHILE` command can be continued to show logical structure.

### Example

```
IF $name = "Washington"  
    DISPLAY (0,0) $name | " - President"  
ELSE  
BEGIN  
    DISPLAY (0,0) $name | " - Vice President"  
    GOTO find_name  
END  
.  
.  
.  
WHILE %time < 10  
    DISPLAY (0,0) "Please wait."
```

If an `IF`, `ELSE` or `WHILE` command contains no command list, commands on the following line are assumed to be a continuation of the line.

---

## Comments

---

Comments are non-executing statements that assist the reader of a script source file in understanding its purpose and implementation. All characters on a line following a “;” (semicolon) are considered to be part of a comment and are ignored by the compiler.

### Example

```
;the following routine displays all phone number records
;
RECORD READ 0 AT 0      ;read first record
WHILE NOT EOF ()      ;stop if end of file
  BEGIN
    DISPLAY @R0.1
    RECORD READ 0      ;read next record
  END
```

Since comments are ignored by the compiler, they are optional; however, the use of comments is recommended as a simple and straightforward way to document a script.



---

## Labels

---

Labels are non-executing statements that mark a location in a script to which the execution of a script can branch. Labels allow you to employ structured programming techniques to create modular scripts (scripts with subroutines). Scripts composed of modules tend to be easier to expand and debug, since each module typically has a single purpose, single entry point, and single exit point.

Labels appear on their own line in a script and consist of an asterisk (\*) followed by a series of characters (alphanumeric characters and underscores), up to 32 characters long. A label cannot contain any spaces, and is not case sensitive. If a label is the name of a subroutine, you can also include a list of arguments after the label. However, the compiler ignores any other statements following a label on the same line.



Also see: [Parameter Passing](#) in this chapter

### Example 1

```
#SecondaryNumber=False
*ConnectService

Set PhoneNumber = "555-1221"
%NumTries=0
Dial

While(not Connect())
Begin
  Increment %NumTries
  If %NumTries = 5
  Begin
    If SecondaryNumber
    Begin
      Dialog
        Message "Unable to Connect"
        Message "Script terminating"
      Dialog End
      Wait Delay "2"
      Cancel
    End
    Dialog
      Message "TR1 - No Connection"
      Message "Trying Secondary Number"
    Dialog End
    Wait Delay "2"
    Perform SetNextNumber
  End
End
.
.
.
*SetNextNumber
#SecondaryNumber=True
Set PhoneNumber = "555-1212"
Goto ConnectService
Return
```

---

## Labels, *continued*

---

The script segment in Example 1 performs a loop. If it fails to connect, it calls the `SetNextNumber` subroutine to attempt a second number. If the second attempt to connect is unsuccessful, the script displays an error dialog and then ends script execution.

### Example 2

```
;Main Routine
;The variables $String, %Int, and !Real are defined in
;this section of the script.
.
.
.
Perform SubRoutine ($String, %Int, !Real)
;The Main Routine starts a subroutine and gives default
;values to all or some of the variables of the subroutine
;with the list of variables between the parenthesis.
;the rest of the main routine
.
.
.
Cancel          ;End of the script

*SubRoutine ($Name, %NumHours, !AvgDaily)
;a subroutine Initially the variable $Name has the same
;value as $String, %NumHours has the same value as %Int,
;and !AvgDaily has the same value as !Real.
.
.
.
;At the end of the subroutine, the subroutine will
;replace the contents of $String, %Int, and !Real with
;the contents of $Name,%NumHours, or !AvgDaily,
;respectively.
Return          ;resume execution after Perform
```

Example 2 contains a module, or subroutine. A standard subroutine usually has:

- ▼ one entry point (the label, which may or may not have arguments), and
- ▼ one exit point (usually the `RETURN` command).

If there are variables in the main routine of the script that the subroutine should act on, the `PERFORM` command should explicitly pass the variables to the subroutine by including both the name of the subroutine (the label) and a list of the variables the subroutine should act upon.

After the subroutine finishes its tasks, the main routine continues at the command following the `PERFORM` command which started the subroutine. Because the contents of the subroutine variables (`$Name`, `%NumHours`, and `!AvgDaily`) are the same as the main routine variables (`$String`, `%Int`, and `!Real`), when the subroutine returns, the main routine variables take on the value of the subroutine variables.

---

# Strings

---

A string can be a string function, a string variable, or a string constant. Complex strings and string expressions are created by joining two or more strings with valid operators.

## String Functions

A string function is simply a function that returns a string. Refer to the list of string functions in **Appendix C Quick Reference**.

## String Constants

A string constant consists of a series of characters enclosed within single or double quotation marks, e.g., 'string' or "string." String constants may contain a maximum of 254 characters.

To embed a quotation mark in a string, alternate the use of single and double quotation marks, e.g., "Arnold's car" or '^\$P "routine\_a". ASCII control characters can be embedded in a string by including a backslash (\) and a three-digit octal code, or a caret (^) and an ASCII character. An ESCAPE can be represented by either "\033" or "^[". A carriage return can be represented by either "\015" or "^M".

A null string is written as double quotes with nothing between them (""). The quotation marks do not appear when a null string is displayed.

## Named String Variables

A named string variable consists of a dollar sign (\$) followed by a variable name. Variable names may contain a maximum of 32 alphanumeric characters or underscores (for example, \$first\_name) and are not case sensitive. As with string constants, a string variable may contain a maximum of 254 characters.

Variables can be referenced indirectly by using another variable to represent the variable name. For example, if \$day is an existing variable and contains the value Monday, creating the variable \$\$day creates the variable \$Monday.

In addition to the named string variable, several system variables are available to represent specific types of string data.

## Record Buffer Variables

A record buffer variable is a string that contains a record. A record is a sub-unit of a table. A table is a structure into which a file can be loaded. Contents of the file can then be manipulated. Similarly, a table collects and organizes masses of data into a single location. The data is then saved to a file. A single record buffer variable may have a maximum of 254 characters.

Contents of a table record cannot be manipulated directly. Record manipulation is performed with the RECORD WRITE and RECORD READ commands to place data into and to retrieve data from a table, respectively. When DCS reads from a table, it places the contents of the table record into the record buffer variable for that table, and when DCS writes to a table, it places the contents of the table's record buffer variable into a record in the table. The record buffer variable is a preparation area where you manipulate the contents of a table record. Each table defined by a script has its own record buffer variable.

---

## Strings, *continued*

---

DCS has two types of tables:

- ▼ Text tables which have variable-length records, and
- ▼ Structured tables which have fixed-length records composed of fields.

The syntax of the record buffer variable depends on the type of table with which it is associated. Record buffer variables have the following syntax:

```
@RTable          ;A record buffer variable for a text table.  
@RTable.Field    ;A record buffer variable for a structured table.
```

The **Table** argument is an integer indicating the record buffer variable for a table defined in a script. DCS can have up to 16 tables defined at one time, and the **Table** argument must be an integer from 0 to 15. The optional **Field** argument is an integer identifying a subdivision of a record buffer variable for a structured table. A structured table may have a maximum of 255 fields, and the **Field** argument must be an integer from 1 to 255.

DCS treats the record buffer variable in a text table similarly to a named string variable. The variable only contains the characters that have been assigned to it, and the number of characters in the variable changes with the characters assigned to it.

Consider the following assignments:

```
@R0 = "This is text"  
;The variable contains 12 characters including two  
;spaces.  
  
@R0 = "Numbers: 12345"  
;The variable now contains 14 characters including  
;one space.  
  
@R0 = @R0 | "More characters"  
;The variable now contains its previous 14  
;characters plus 15 more.
```

These examples illustrate that the record buffer variable for a text table only contains the characters that were assigned to the variable.

DCS handles the record buffer variables for structured tables differently from text tables, because it composes structured table records of fields. Table records are similar to rows in a spreadsheet, and the record fields are similar to the cells in the rows. In a structured table each field must be associated with a data type and size. The data types for fields in a structured table are character (CHAR), real numbers (REAL), and integer numbers (INT). Because you determine the size of each field, the records in a structured table are always the same size.

All record buffer variables and any fields within them are strings. However, a script defines the fields of a structured table record with the **TABLE DEFINE** command. Each field can be one of the following types: character, integer, or real.

---

## Strings, *continued*

---

For example, the TABLE DEFINE command might appear in a script as follows:

```
TABLE DEFINE 0 FIELDS CHAR 10 REAL 6 INT 9
```

The keywords and numbers after the word `FIELDS` delineate the fields of the record for the table numbered zero. Each record in the table defined above has three fields, and the record buffer variable for this table has the same structure. The keywords `CHAR`, `REAL`, and `INT` indicate the logical data type of the field (characters, real numbers, and integer numbers, respectively). These types are a part of the definition of a structured table to allow you to deal with the data in a field in a logical fashion. The numbers after the keywords indicate the size of the fields. The first field contains ten characters, the second field contains six characters, and the third field contains nine characters. The size of the fields are the same no matter what you have assigned to the fields. However, you may define a field with space for a maximum of 254 characters.

If the data you assign to a field in a record does not fill the field, DCS pads the unassigned positions with spaces. The fields defined as character data are left justified. For example, consider the following assignments:

```
TABLE DEFINE 0 FIELDS CHAR 10 REAL 6 INT 9
;creates structured table zero

@R0.1 = "Numbers: 12345"
;The variable contains ten characters: Numbers: 1

@R0.1 = "Text"
;The variable contains ten characters: the word Text
;and six trailing spaces

@R0.1 = @R0 | "More characters"
;The variable contains the previous ten characters:
;Text and six trailing spaces

@R0.1 = TRIM (@R0) | "More characters"
;The TRIM function deletes the trailing spaces and
;the variable now contains another set of ten
;characters: TextMore c
```

However, if a field has a numeric definition (`INT` or `REAL`), DCS right justifies the data in the field and pads the unassigned positions with preceding spaces. If the numbers placed in a field have more digits than a field has positions, DCS truncates the rightmost, or least significant, digits.

---

## Strings, *continued*

---

Consider the following examples:

```
@R0.2 = "1.234"  
;The variable contains six characters: one preceding  
;space and 1.234  
  
!RealNum = Num (@R0.2) + Num (@R0.2)  
;The NUM function transforms the string into numeric  
;values. The result is 2.468 and is placed in the  
;variable !RealNum  
  
@R0.2 = "67.54321"  
;The variable contains six characters: 67.543  
;The characters 2 and 1 are truncated  
  
@R0.3 = "12345"  
;The variable contains nine characters: four spaces  
;and 12345  
  
@R0.3 = "998765432194"  
;The variable contains nine characters: 998765432.  
;The characters 194 are truncated  
  
!RealNum = Num (@R0.2) + Num (@R0.3)  
;The NUM function transforms the strings into  
;numeric values. The result is 998765499.543 and is  
;placed in the variable !RealNum
```

### Using Record Buffer Variables

This command creates structured Table 6. It has three fields.

```
TABLE DEFINE 6 FIELDS CHAR 10 INT 3 INT 5  
@R6 = "test      123   38"
```

The record buffer variable for this table is @R6. To access each field of Table 6 directly, the following record buffer variables are possible:

```
@R6.1 accesses the first field (CHAR 10),  
@R6.2 accesses the second field (INT 3), and  
@R6.3 accesses the third field (INT 5).
```

To access all of the fields of @R6, do not include the period and the field number; rather, include a string of length equal to the combined lengths of the fields. If the data in a field contains fewer characters than the defined field length, insert space characters to the right or the left of the data to fill the field to its exact length. This ensures that the correct data is placed into the correct field.

---

## Strings, *continued*

---

In the example on the previous page, @R6 is treated like an 18-character string (the fields CHAR 10, INT 3, and INT 5 have a combined length of 18):

- ▼ The data in field CHAR 10, test, contains only four characters, thus six spaces are added to the right of test;
- ▼ The data in field INT 3, 123, contains three characters, thus no spaces are added; and
- ▼ The data in field INT 5, 38, contains two characters, thus three spaces are added to the left of 38.

These commands display the entire first record of Table 6 in the terminal window:

```
RECORD READ 6 AT 0
;read table six at the first record
DISPLAY @R6
```

These commands modify the contents of Table 6:

```
@R6.2 = "55"
RECORD WRITE 6 AT 2
```

The first command assigns the character string 55 to the second field of the record buffer variable for Table 6. The second command writes all of the current fields in the record buffer variable to table six and in the third record position. The command phrase AT 2 is the third record position, because the record positions start at 0 (zero).

DCS allows a script to use integer variables for the **Table** and **Field** arguments, when referring to record buffer variables.

### Example

These commands create a loop which sequentially displays each field of the current record.

```
%TableNum = 7
TABLE DEFINE %TableNum FIELDS INT 3 CHAR 10 CHAR 10
TABLE LOAD %TableNum "DATA.DCM" AS TEXT
;Places the contents of the file into the table.
RECORD READ %TableNum
%Field = 1
WHILE %field <= 3
BEGIN
    DISPLAY @R(%TableNum).%field | "^M^J"
    INCREMENT %field
END
```



Also see: **Tables, Records, and Data Manipulation** in this chapter

---

## Strings, *continued*

---

### Function Key and Function Key Title Variables

A *function key variable* accesses the string contents of the command portion of a function key. A function key command may have a maximum of 42 characters. The variable observes the following syntax:

```
@Fkey
```

The **Key** argument is a numeric specifying the number of the function key. Since eight function keys on each of four levels may be defined, the **Key** argument must take on a value from 1 to 8. The **Key** argument may also be specified by a numeric variable containing a valid value. The function key level is specified using the **LEVEL** command.



Also see: **LEVEL** command

The *function key title variable* accesses the string title of a function key. The title of a function key can have a maximum of 18 alphanumeric characters. Function key title variables observe the following syntax:

```
@Tkey
```

The **Key** argument is a numeric specifying the number of the function key. DCS allows the creation of eight function keys (numbered 1 through 8) on each of four levels. The **Key** argument may also be specified by a numeric variable containing a valid value. DCS also allows you to modify the title of the “Level: n” key, which changes the function key level. Thus the **Key** argument must have a value from 1 to 9, with 9 indicating a title change for the “Level: n” key.

```
@T9 = "Next Level"  
;Changes the title of the Level button.
```

```
@T9 = ""  
;Changes the Level button title to its default.
```

Function key and function key title variables may be used as arguments in all functions and commands that accept string variables.



---

## Strings, *continued*

---

### Example

These commands:

```
LEVEL 1
@T1 = "NUMBER"
@F1 = "17135552334"
@T2 = "PASSWORD"
@F2 = "mysecretcode"
LEVEL 2
@T%num = "MAIL SYSTEM"
@F%num = "MAIL^M"
```

define two titled function keys on Level 1 and one key on Level 2:

- ▼ The first key on Level 1 has the title `NUMBER` and contains the string `"17135552334"`, which represents a phone number.
- ▼ The second key on Level 1 has the title `PASSWORD` and contains the string `"mysecretcode"`.
- ▼ For Level 2, only the key specified by the integer variable `%num` is defined. In this case, the value of `%num` must be within the valid range for the title and function key variables.

A function key command can specify a string to send to the remote system or specify a special function. Strings can include valid META keys for the connected session as well as the special line control strings (^M and ^J). The following special functions correspond to commands in the DCS script language:

```
^$B = BREAK
^$E = EXECUTE
^$L = LEVEL
^$P = PERFORM
```

The `BREAK`, `EXECUTE`, `LEVEL`, and `PERFORM` commands have analogous commands in the script language. `PERFORM` and `EXECUTE` can pass parameters in the script language, but not from a function key.

Also, the above `PERFORM` and `EXECUTE` commands must use the *far target* addressing form for a script subroutine. A copy of the script with the desired subroutine will be loaded and executed to support the selected function key. The syntax specified in this DCS Script Language Reference should be used for these commands.

---

## Strings, *continued*

---

### Settings Variables

The Settings variable is similar to the named string variable, except that it is stored as part of a session file, and its contents depend on the loaded session file. A Settings variable may hold a maximum of 42 characters. A Settings variable observes the following syntax:

```
@Sn
```

The *n* argument specifies the desired settings variable number. Since eight settings variables may be defined, the *n* argument must take on a value from 1 to 8. The *n* argument may also be specified by a numeric variable containing a valid value.

If a settings variable is saved in a session file, the contents of that variable are restored the next time the session file is loaded. If the value is changed during script execution, then the session file must be saved for the settings variable to retain the new value.

Settings variables may be used as arguments in all functions and commands that accept string variables.

#### Example 1

These scripts:

##### Script 1

```
@S1 = "13121234567"  
@S2 = "password-1"  
SAVE "chicago"
```

##### Script 2

```
@S%idnum = "17131234567"  
@S%pass = "password-2"  
SAVE "houston"
```

assign values to the settings variables @S1 and @S2. Those values are then saved in the specified session files. In the second script, the setting variable numbers are specified by integer variables. In this case, the values of %idnum and %pass must be within the valid range for setting variables.

---

## Strings, *continued*

---

### Example 2

This script:

```
DIALOG
    EDITTEXT "Enter City"
    BUTTON DEFAULT "OK" RESUME
DIALOG END
WAIT RESUME

IF EXISTS (Directory (Settings) | EDITTEXT (1))
BEGIN
    LOAD EDITTEXT (1)
END
ELSE
    CANCEL
DIALOG CANCEL
DIAL @S1
WAIT STRING "Password"
SEND @S2 | "^M"
```

instructs DCS to dial a phone number and send a password string. The dialog box determines which city the user wants to call. A `LOAD` command loads the appropriate session file for the city chosen. The `DIAL` and `SEND` commands then use the values appropriate for that city.

### String Expressions

String expressions containing multiple operands are created by joining two or more strings with the concatenation operator (`|`). String expressions differ from complex strings only in that they are not enclosed in parentheses and cannot be used as arguments where a single string operand is expected.

#### Example

```
$date = $month | " " | $day | ", " | $year
```

If the string variables `$month`, `$day`, and `$year` contain the values January, 8, and 2001, respectively, the string variable `$date` contains the value "January 8, 2001".

A maximum of 254 characters can be placed in a string variable. If the number of concatenated characters exceeds 254, additional characters are ignored.

---

## Strings, *continued*

---

### Complex Strings

A complex string consists of multiple string operands joined by the concatenation operator (`()`), and enclosed in parentheses. A complex string may be used as an argument where a single string operand is expected.

#### Example

In this example:

```
%length = LENGTH ($firstname | " " | $lastname)
```

if the string variable `$firstname` contains “George”, and the string variable `$lastname` contains Washington, the numeric variable `%length` contains the value 17 (the space between each string is included).

---

## Numerics

---

A numeric operand consists of a numeric function, a numeric variable, or a numeric constant preceded by an optional modifier. Complex numerics and numeric expressions are created by joining two or more numeric operands with valid operators. DCS supports both integer and real numeric operands.

### Numeric Functions

A numeric function returns a value that is a numeric. Refer to the grouping of numeric functions in **Appendix C Quick Reference** for a list.

### Numeric Constants

A numeric constant consists of an integer or real number, such as 25 or 3.14. Decimal numerics are represented as `n` (for example, `234`). Hexadecimal numerics are represented as `0xn` (for example, `0x3D`). Octal numbers are represented as `\nnn` (for example, `\033`).

### Named Numeric Variables

An integer numeric variable consists of a percent sign (`%`) followed by a variable name. A real numeric variable consists of an exclamation point (`!`) followed by a variable name. Variable names can contain up to 32 alphanumeric characters or underscores (for example, `%bytes`, `!cash`), and are not case sensitive. You may reference variables indirectly by using another variable to represent the variable name. For example, if `%order` is an existing variable, and contains the value 1, creating the variable `!%order` creates the variable `!1`.

There is one valid numeric modifier: the unary minus. It converts a numeric to its opposite value, and is equivalent to multiplying the numeric by -1 (negative one).

### Example

```
-%num
```

If the numeric variable `%num` contains -23, `-%num` evaluates to 23.

It is possible to convert a numeric type from an integer to a real, or from a real to an integer. When a real value is converted to an integer value, all decimal places are truncated, not rounded. Values can be converted using the `REAL` and `INT` functions, or by directly assigning a real or integer value to the desired variable type.

---

## Numerics, *continued*

---

### Numeric Expressions

Numeric expressions are created by joining two or more numerics with a valid operator (+, -, \*, /, %). Numeric expressions differ from complex numerics only in that they are not enclosed in parentheses and cannot be used as arguments where a single numeric operand is expected.

#### Example

In this example:

```
%sum = %wheat + %corn + !rice
```

the integer numeric variable %sum is set to the sum of the three numeric variables.

In this example of modulus division:

```
%remain = 25 % 7
```

the integer variable %remain is set to 4, the remainder of the modulus division operation.

### Complex Numeric Variables

A complex numeric variable consists of multiple numeric operands joined by a valid operator. The valid numeric operators are: +, -, \*, /, %. These operators correspond to the operations of addition, subtraction, multiplication, division, and modulus. The multiplication, division, and modulus operators take precedence over the addition and subtraction operators, but parentheses can be used to alter this precedence. The modulus operation is not defined for real numerics. The result of all numeric operations is a real numeric, but can be converted to an integer either by using the INT function, or by storing the result as an integer numeric variable.

#### Example 1

In this example:

```
%integer = 10/3
```

even though the result of the operation 10/3 is a real value, this command stores the result in an integer numeric variable. The real result is, therefore, truncated and the integer numeric variable contains the value 3.

In this example:

```
!percentage = 10/3
```

the real numeric variable !percentage contains the value 3.333333333333333. Real numerics precise to the 15th decimal place.

---

## Numerics, *continued*

---

A complex numeric variable may be used as an argument where a single numeric operand is expected.

In this example:

```
%number = RANDOM (%range - 1)
```

if the integer variable `%range` contains the value 10, `%number` contains a random number generated in the range from 0 to 8 inclusive.

---

## Booleans

---

A Boolean may be a Boolean function, a Boolean variable, a Boolean constant, or a relational expression with an optional modifier. Complex Boolean and Boolean expressions are created by joining two or more Boolean operands with valid operators.

### Boolean Functions

A Boolean function returns the value TRUE or FALSE. Examples of Boolean functions are:

- ▼ CONNECT
- ▼ ERROR
- ▼ ZOOMED

Refer to the Boolean Functions group in **Chapter 2 Functions** for a list of all Boolean functions.

### Boolean Constants

A Boolean constant is specified by the keyword TRUE or FALSE. The keyword is not enclosed within quotation marks as in the case of a string constant.

### Named Boolean Variables

A Boolean variable consists of a number sign (#) followed by a variable name. Variable names can contain up to 32 alphanumeric characters or underscores (for example, #state) and are not case sensitive. Variables can be referenced indirectly by using another variable to represent the variable name. For example, if %x is an existing variable and contains the value 5, creating the variable #%x creates the Boolean variable #5.

### Relational Expressions

A relational expression is created by comparing two strings or two numerics using one of the following relational operators:

Operator	Relationship
<	Less than
>	Greater than
=	Equal to
==	Equal to
◇	Not equal to
!=	Not equal to
<=	Less than or equal to
>=	Greater than or equal to

There is one valid Boolean modifier: NOT. It converts a Boolean to its logical opposite.



---

## Booleans, *continued*

---

### Example

```
IF %income > %net
    DISPLAY (0, 0) STR (%income - %net) | "dollars earned"
ELSE
    DISPLAY (0, 0) "No profit made this quarter"
```

In this example, `%income > %net` is the relational expression evaluated. If it evaluates to TRUE, the amount of profit is displayed. If it evaluates to FALSE, the message "No profit made this quarter" is displayed.

### Boolean Expressions

Boolean expressions containing multiple operands are created using the Boolean operators AND and OR. NOT may also be used as an operator and has precedence over AND and OR. Boolean expressions differ from complex Booleans in that they are not enclosed in parentheses and may not be used as arguments where a single Boolean operand is expected. There are no functions or commands that expect only single Boolean operands; therefore, complex Booleans and Boolean expressions may be used interchangeably.

### Example

This Boolean expression:

```
NOT EOF ()
```

evaluates to TRUE if the EOF function returns FALSE.

This Boolean expression:

```
$state <> "TEXAS"
```

evaluates to TRUE if `$state` contains a string other than TEXAS.

This Boolean expression:

```
%num <= 100
```

evaluates to TRUE if `%num` contains a numeric value less than or equal to 100.

This Boolean expression:

```
$state <> "TEXAS" AND $state <> "FLORIDA"
```

evaluates to TRUE if `$state` contains any string other than TEXAS or FLORIDA.

This Boolean expression:

```
%1 < 25 OR %1 > 50
```

evaluates to TRUE if `%1` contains any numeric value other than those within the range 25-50, inclusive.

---

## Booleans, *continued*

---

### Complex Booleans

A complex Boolean consists of multiple Boolean operands joined by a valid operator. The valid Boolean operators are AND and OR. A complex Boolean consisting of operands joined by AND is TRUE if all associated operands are TRUE. A complex Boolean consisting of operands joined by OR is TRUE if any inclusive associated operands are TRUE. The AND operator takes precedence over the OR operator, but parentheses can be used to alter this precedence.

### Example

This Boolean expression:

```
(%1 < 100 AND %2 < 100) OR ($state = "FLORIDA")
```

evaluates to TRUE if both %1 and %2 contain numeric values that are less than 100, or \$state contains the string "FLORIDA", or if both statements are true.

---

## Special Argument Types

---

This section refers to several forms of the defined argument types. These argument types appear frequently as arguments for the commands and functions defined in **Chapter 2 Functions** and **Chapter 3 Commands**.

### File Names

File name arguments are string arguments used to specify any file name acceptable to the operating system. File name arguments observe the following syntax:

**Filename = drive: \path\ path...\ filename.ext**

File name arguments can be further specified as either Source or Destination arguments. A Source argument specifies a file name containing the data to be exported. A Destination argument specifies a file name into which data is imported. Source and Destination arguments observe the following syntax:

**Source = drive: \path\ path...\ filename.ext**

**Destination = drive: \path\ path...\ filename.ext**

### Path and File Names

Path and file names may be represented in the DCS Script Language by:

- ▼ String constants
- ▼ String variables
- ▼ String expressions
- ▼ Complex strings

A sequence of characters consisting of a backslash (\) followed by three octal digits is interpreted as a control character. Therefore, you must be careful when specifying path and file names that begin with numerals. To specify a path that begins with three numerals, a double backslash (\\) followed by the three numerals must be used.

For example, you would specify:

```
LAUNCH "G:\\123W\\123W.EXE"
```

to launch Lotus 1-2-3 from G:\123W\123W.EXE.

### Table Arguments

Table arguments are numeric arguments used to specify a table number. Table arguments observe the following syntax:

**%Table = Numeric**

The Numeric argument must be a numeric operand (constant, variable, expression, or function) from 0 to 15, corresponding to one of the available table structures. For more information about tables, see **Tables, Records, and Data Manipulation** in this chapter.

---

## Special Argument Types, *continued*

---

Target arguments are used to reference a label in an executable script file. A target argument can be either a:

- ▼ Near target
- ▼ Far target

### Near Targets

A near target references a label within the current script. The near target argument is a set of characters that specify the desired label. The characters are not enclosed in quotation marks as in the case of a string constant. Near target arguments use the following syntax:


COMMAND Label

where COMMAND is one of the following commands:

- ▼ EXECUTE
- ▼ PERFORM
- ▼ GOTO

and where Label is a string.

Near targets are resolved at compile time, and therefore execute faster than far targets. A syntax error occurs during compilation if the designated near target is not defined in the script.

 **Note:** When a PERFORM or EXECUTE command is assigned to a function key, the target must always be in far target format and must include both the name of the script and the label (even if the label is within the same script as the KEY command).

 Also see: **Function Key Variables** in this chapter

KEY command

---

## Special Argument Types, *continued*

---

### Example

In this example:

```
*ReadLoop
RECORD READ 0 at 0

;beginning of loop
While not EOF ()
Begin
    Display @R0
    Record Read 0
End
Perform end_read
.
.
.
Cancel

*end_read
Display "End of table reached"
Return
;end of script execution
```

near targets are used as arguments for the **PERFORM** command. A record is read from Table 0 (zero). It then displays that record in the terminal window. When the end of file is reached, the EOF function returns TRUE and execution branches to the line labeled `*end_read`, where a closing message is printed and script execution is terminated.

### Far Targets

Far targets are resolved during execution, so they can reference a label within the current script or within another script. The far target argument is a string expression specifying an optional script name and label. If the designated far target is undefined, a critical error is generated during script execution. Far target arguments observe the following syntax:

```
COMMAND "Script*Label"
```

where **COMMAND** is one of the following commands:

- ▼ EXECUTE
- ▼ PERFORM
- ▼ GOTO

where **Script** is the full file name of a compiled script, and


where **\*Label** is a label found in **Script**.

A far target must include at least one of the two arguments, except as noted below. If the **Script** argument is not included, DCS defaults to the current script. If the **Label** argument is not included, execution begins at the first line in the specified script.

---

## Targets, *continued*

---

 **Note:** In cases where you wish to assign a `PERFORM` or `EXECUTE` command to a function key (see both function key variables and the `KEY` command), the target must always be in far target format, and must include both the name of the script and the label (even if the label is within the same script as the `KEY` command).

### Example

```
*readloop
RECORD READ 0 at 0

;Beginning of Loop
While not EOF ()
Begin
    Display @R0
    Record Read 0
End

Display "End of table reached"
Perform "Dialog*ReadComplete"
Cancel
;end of script execution
```

This script accomplishes the same task as the near target example, but the target is now a message box from another script. A record is read from Table 0 and then displayed in the terminal window. If the end of the table is reached, the `EOF` function returns `TRUE`, and displays a closing message. Execution then branches to the line labeled `ReadComplete` in the script `Dialog` and upon return this script ends.

### Example

In this example:

```
WHILE NOT EOF ()
BEGIN
    RECORD READ 0
    IF ERROR ()
        GOTO error ;error is a near target
END
PERFORM "table.dct" ;call another script
CANCEL ;end script after executing "TABLE.DCT"

*error
DISPLAY "Error Encountered ^M"
CANCEL
```

all records from Table 0 (zero) are read and then the script "TABLE.DCT" is performed. If an error occurs during the record read, execution branches to the line labeled `*error`.

---

## Variable Creation

---

Numeric variables, Boolean variables, and string variables (other than the system variables @R, @F, @T, @S, ConnectMessage, ConnectResult, DefaultSessionHandle, NetID, Password, Phonenummer, Result, and UserID) must be created (assigned a value) before they are used.

### Direct Variables

A command that assigns an initial value to a variable is considered to have created that variable. Several commands can create variables:

- ▼ SET
- ▼ ARGUMENTS
- ▼ COLLECT
- ▼ PARSE
- ▼ FILE CREATENAME
- ▼ FILE OPENNAME
- ▼ (DDE) REQUEST
- ▼ WHEN ECHO
- ▼ WHEN INPUT

The assignment operator (=) may also be used to assign an initial value to a variable. It assigns the value on the right of the equal sign to the variable to the left of the equal sign:

```
$name = "Lucy"
```

String variables are created using any of the above commands. Numeric and Boolean variables are created using:

- ▼ Assignment operator (=)
- ▼ ARGUMENTS command
- ▼ SET command

All other commands and functions using variables as arguments must use variables that were previously created using one of the above commands.

### Examples

In this example:

```
COLLECT $DataIn  
DISPLAY $DataIn
```

the variable \$DataIn is created during the execution of the COLLECT command, and can subsequently be used by the DISPLAY command.

---

## Variable Creation, *continued*

---

This script:

```
Record Read 0 At 0
While not EOF ()
Begin
    Increment %count
    Display "Record " | STR (%count) | " = " | @R0 | "^M"
Record Read 0
End
```

causes an execution error because `%count` was not created before being used as an argument for the `INCREMENT` command. To correct this error, `%count` must be given an initial value by a `SET` command or the assignment operator. The following script corrects the error.

In this script:

```
%count = 0
Record Read 0 AT 0
While not EOF ()
Begin
    Increment %count
    Display "Record " | STR (%count) | " = " | @R0 \
        | "^M"
Record Read 0
End
```

because the `INCREMENT` command does not create variables, this script corrects the previous script by creating the variable `%count` using the assignment operator (`=`) before it is used by the `INCREMENT` command.

### Indirect Variables

Variable creation can also be accomplished by using existing variables to create new ones. These are known as indirect variables.

Indirect variable names are created using integer (`%`) or string (`$`) variable types. This is done by placing the variable type signifier at the beginning of the existing variable. For example, if `$varname="x"` and `;%$varname=1`, then the value of `%x` will be 1.

String and integer numeric variables may be used to create indirect variable names for string, numeric and Boolean variables. A Boolean variable cannot be used to create another variable. Integer numeric variables may also be used to create indirect variables for system, function key titles, function key assignments, and record variables. For example, if `%index=3` and `@s(%index="secret"`, then `@s3` will be assigned the string `secret`.

Indirect variables provide flexibility in coding a script. A change in the value of an indirect variable is reflected throughout the code, thus changing any variable with which it is associated. This can make it easier to maintaining a script, or allow the script to respond to input from different users.



---

## Symbols

---

Symbols are characters other than alphabetical or numerical characters (punctuation, for example) which are used to indicate a special condition or to indicate a particular variable type. Many symbols also function as operators.

### Symbols

Available symbols include:

* (asterisk)	:	(colon)	# (number sign)	?	(question mark)	
@ (at sign)	,	(comma)	() (parentheses)	"	(quote, double)	
\ (backslash)	\$	(dollar sign)	% (percent sign)	'	(quote, single)	
^ (caret)	!	(exclamation point)	.	(period)	;	(semicolon)

### \* (asterisk)

Asterisks signal a label. Labels indicate subsections and subroutines in a script.

### Examples

In this example:

```
IF ERROR ()
BEGIN
    PERFORM Error
END
DISPLAY "OK" | "^M"
CANCEL

*Error
DISPLAY "Error" | "^M"
CANCEL
```

execution branches to `*error` and displays “Error” if the `ERROR` function is set to `TRUE`; otherwise, “OK” is displayed, and script execution is terminated.

Asterisks are also the default wildcard character used to represent 0 (zero) or more arbitrary characters. In this example:

```
%Row = 0
$File = Route (directory(task) | "*.DCP")
While not Error ()
Begin
    Display (%Row, 0) $File | "^M"
    Increment %Row
    $File = Next ()
End
```

all script source files in the current working directory are displayed. The wildcard (\*) used in the path argument of the `ROUTE` function is used to select any file that has the DCP (\*.dcp) extension.

---

## Symbols, *continued*

---

### @ (at sign)

“At” signs are used to indicate specific system string variables.

#### Examples

```
@R6.2 = "Tuesday"  
;Record Buffer Variable
```

```
@F3 = "555-2327"  
;Function Key Variable
```

```
@T3 = "Number"  
;Function Key Title Variable
```

```
@S4 = "Remote"  
;Settings Variable
```

### \ (backslash)

Backslashes are used to:

- ▼ Continue a command from one line to the next:

```
TABLE DEFINE 0 FIELDS CHAR 10 \  
CHAR 20 INT 15 INT 3 INT 15
```

- ▼ Indicate octal numerics:

```
%octal = \067
```

- ▼ Separate drive designations, directories, and file names in DOS path names:

```
$path = "C:\DCSERIES\SCRIPTS\  
$file = ROUTE ($path | "*.*)"
```

### ^ (caret)

Carets are used to embed ASCII control characters in a string.

In this example:

```
@F3 = "^[ LOGON ^M"
```

carets are used to assign a string containing an escape and a carriage return to a function key.

### : (colon)

Colons are used after the drive letter designation in DOS path names:

```
$path = ROUTE ("C:\DCSERIES\*.*)"
```

---

## Symbols, *continued*

---

### , (comma)

Commas are used to:

- ▼ Continue a command block from one line to the next without using a BEGIN and END pair:

```
IF $date = "03/25/94"  
    SET $month "March",  
    SET $day "25",  
    SET $year "1994"
```

- ▼ Separate parameter lists in function calls.

```
$Last = substr ($last, 1, 1)
```

### \$ (dollar sign)

Dollar signs indicate a named string variable:

```
$state = "Texas"
```

### ! (exclamation point)

Exclamation points indicate a real numeric variable:

```
!pi = 3.1415926535897
```

### # (number sign)

Number signs (or pound signs) are used to indicate Boolean variables:

```
#selected = TRUE
```

### () (parentheses)

Parentheses are used to:

- ▼ Enclose function arguments as in this example:

```
SEARCH (0, 3, 20, "Texas")
```

- ▼ Control the order in which an expression is evaluated:

```
!num1 = 3 + 4 / 10 + 8 * 5  
!num2 = ((3 + 4) / 10 + 8) * 5
```

In this example, !num1 evaluates to 43.4, but !num2 evaluates to 43.5, due to the use of parentheses to change the order of evaluation.

---

## Symbols, *continued*

---

### **% (percent sign)**

Percent signs indicate an integer numeric variable:

```
%row = 3
```

### **. (period)**

Periods indicate the decimal place in a (decimal) real numeric operand:

```
!number = 342.77
```

Periods are also used as DOS file extension separators:

```
$file = ROUTE ("LOGON.DCT")
```

Periods also specify a table field in a record buffer variable:

```
@R12.4 = "1001"
```

### **? (question mark)**

Question marks are the default wildcard character used to represent any single arbitrary character:

```
%Row = 0
$File = Route (directory(task) | "*.DC?")
While not Error ( )
Begin
    Display(%Row, 0) $File | "^M"
    Increment %Row
    $File = Next ( )
End
```

The ? wildcard used in the path argument of the ROUTE function is used to select any file whose extension begins with "DC" and is three letters in length.

### **" (quotation mark, double), ' (quotation mark, single)**

Both single and double quotation marks are used to enclose strings. To use quotation marks within a string, alternate single and double quotes:

```
$state = "Texas"
```

In this example, \$state contains the characters Texas.

```
$quote1 = ``Y'all come and see us!``
$quote2 = `` | Y'all come and see us! | ``
```

In this example, the script line containing \$quote1 would not compile, since the single and double quotes are unbalanced. The characters ``Y (a double quote and a capital Y) are enclosed by two single quotes; however, the string all come and see us! ends with a double and single quote. On the other hand, the script line containing \$quote2 will compile, and \$quote2 will contain a string that is composed of the alphabetic characters and single quote enclosed in

---

## Symbols, *continued*

---

double quotes (“Y’all come and see us!”). In the script line, the double quotes are enclosed by single quotes and joined with the main string by concatenation.

### **;** (semicolon)

Semicolons begin a comment (useful for internally documenting a script). All characters on a line following a semicolon are ignored by the compiler:

```
;Script File Finder  
;  
$file = ROUTE (“*.DC?”)  
;returns name of file
```

---

## Operators

---

Operators are either a single character or a group of characters that represents arithmetic, string or logical operations on the operands within an expression; plus the assignment operator, which creates or modifies a variable and is equivalent to the SET command.

Available operation symbols and Boolean syntax include:

<code>+</code> add	<code>=</code> assign	<code> </code> concatenate	<code>%</code> modulus
<code>-</code> subtract	<code>AND</code>	<code>==</code> equal to	<code>&lt;&gt;</code> not equal to
<code>*</code> multiply	<code>OR</code>	<code>&gt;</code> greater than	<code>&lt;</code> less than
<code>\</code> divide	<code>NOT</code>	<code>&gt;=</code> greater than or equal to	<code>&lt;=</code> less than or equal to

### **+ (add)**

The plus sign is the addition math operator:

```
%sum = 34 + 12 + 3
```

### **- (subtract)**

The minus sign is the subtraction math operator:

```
%sum = 34 - 12 - 3
```

It can also be used as the unary negative operator:

```
%num = -1
```

### **\* (multiply)**

The asterisk is the multiplication math operator:

```
!result = 27.2 * 18
```

Multiplication operations are evaluated before addition and subtraction operations in an expression.

### **/ (divide)**

The slash is the division math operator:

```
!result = 27.2 / 18
```

Division operations are evaluated before addition and subtraction operations in an expression.

---

## Operators, *continued*

---

### **% (modulus)**

The percent sign is the modulus math operator:

```
%remainder = %num1 % %num2
```

The `%remainder` variable contains the whole number remainder of the division of `%num1` by `%num2`.

Modulus operations are evaluated before addition and subtraction operations in an expression. The modulus operations are defined for whole numbers, or integers, not real numbers.

### **= (assign)**

The equal sign is the assignment operator (equivalent to the SET command):

```
1 SET $target $source
2 $target = $source
```

Line **1** is equivalent to line **2**; both set the variable `$target` to the value of `$source`.

### **= or == (equal to)**

The equal sign is also the equal to relational operator:

```
IF $user = $sysop
    DISPLAY (0, 0) "administrator online ^M"
```

This example displays a message if the string `$user` is equivalent to the string in `$sysop`. You can also use `==` for equal to.

### **< (less than)**

The less than sign is the less than relational operator:

```
IF %sessions < 2
    DISPLAY (0,0) "new user online ^M"
```

This example displays a message if the integer numeric `%sessions` is less than 2.

### **> (greater than)**

The greater than sign is the greater than relational operator:

```
IF SECONDS ($logon) > 3300
    DISPLAY (0,0) \
        "You have less than" | \
        STR ((%session_time - 3300) / 60) \
        | "minutes left ^M"
```

This example displays a message if the elapsed time since the time specified by `$logon` is greater than 55 minutes.

---

## Operators, *continued*

---

### **<= (less than or equal to)**

The less than and equal signs are used together as the less than or equal to relational operator. The order of the two signs is not important (<= and =< will both work):

```
IF %sessions <= 2
    DISPLAY (0,0) "new user online ^M"
```

This example displays a message if the integer numeric `%sessions` is less than or equal to two.

### **>= (greater than or equal to)**

The greater than and equal signs are used together as the greater than or equal to relational operator. The order of the two signs is not important (>= and => will both work):

```
IF SECONDS ($logon) >= 3300
    DISPLAY (0,0) "You have less than" | \
        STR ((%session_time - 3300) / 60) \
        | "minutes left ^M"
```

This example displays a message if the elapsed time since the time specified by `$logon` is greater than or equal to 55 minutes.

### **<> (not equal to) or !=**

The greater than and less than signs are used together as the Not Equal To relational operator. The exclamation point and equal sign can also be used together (!=) as the Not Equal To operator:

```
CONNECT
WAIT QUIET "2"
IF SEARCH ("USERID") <> -1
    SEND "userid{tab}password{enter}"
ELSE
    DISPLAY "USERID prompt not found ^M"
CANCEL
```

In this example, written for an IBM 3270 connection, DCS establishes a connection with the host, waits for two seconds, and then searches the screen for the USERID prompt. If it is found, DCS logs into the system. If it is not found, the message "USERID prompt not found" is displayed and script execution is stopped.



---

## Operators, *continued*

---

### NOT

The **NOT** Boolean modifier converts a Boolean to its logical opposite:

```
RECORD READ 0
WHILE NOT EOF ( )
BEGIN
    DISPLAY @R0
    RECORD READ 0
END
```

This example displays each record in Table 0 until reaching the end of the file.

### AND

The **AND** Boolean operator is used in complex Boolean expressions. The expression evaluates to TRUE if all associated operands are TRUE:

```
TABLE DEFINE 0 FIELDS CHAR 3 INT 4 INT 8
TABLE LOAD 0 FROM "SALES.DCM" AS TEXT
RECORD READ 0
IF @R0.1 = "Jan" AND @R0.2 > "1998"
    SEND @R0
ELSE
    DISPLAY "Old or incomplete file"
CANCEL
```

This example defines a table, and loads the SALES.DCM file into the table. DCS then verifies that the first two fields in the first record match the expected format. If the fields match, the record buffer is sent to the remote system. If they do not match, "Old or incomplete file" displays and script execution stops.

**AND** relations are evaluated before **OR** relations; this precedence may be changed by using parentheses.

### OR

The **OR** Boolean operator is used in complex Boolean expressions. The expression evaluates to TRUE if any of the associated operands are TRUE:

```
IF $password = "newuser" \
OR $password = "" AND \
$userid = "visitor"
DISPLAY (0,0) "new user online"
```

This example displays a message if \$password is newuser or if \$password is a null string and \$userid is visitor.

---

## Operators, *continued*

---

### I (Concatenate)

The vertical bar is the concatenation string operator:

```
IF SECONDS ($logon) > 3300
  DISPLAY (0,0) \
  "You have less than " | \
  STR ((%session_time - 3300) / 60) \
  | "minutes left"
```

In this example, concatenation operators are used to insert a dynamic value into the displayed string. For example, if the integer numeric `%session_time` contains the value 3600, the `STR` function returns the value 5, and displays the string "You have less than 5 minutes left".

### Operator Precedence

The following table lists the precedence for math and relational operators. All operators associate from left to right.

Precedence	Operators
Highest	( )
	NOT      - ( <i>unary minus</i> )
	*          /          %
	+          -
	>          >=      <          <=
	= ( <i>equal to</i> )    ==      <>      !=
	AND
	OR
Lowest	= ( <i>assign</i> )

---

## Scoping Rules

---

DCS Script Language variable management scheme automatically creates variables when they are assigned an initial value and makes them available only to the routine in which they were created. This prevents the script writer from having to explicitly create and dispose of variables. This scheme, however, imposes some restrictions which make it necessary to consider the range over which a variable is defined. Scoping rules define this range.

If script execution branches, a parent is defined as a routine on a level above a child routine. A child routine is created in one of three ways:

- ▼ A child routine can be called with a **PERFORM** command. The routine containing the **PERFORM** command is the parent; the called routine is the child.
- ▼ A routine executed from the command list of a **WHEN** command is considered a child of the routine containing the **WAIT** command it interrupted.
- ▼ A dialog definition is considered a child of the routine that created it. Any script or routine can be a parent to one routine *and* a child of another.

### Example

In this example:

```
*Main
If Connect ()
    $Name = ``*Yes``
Else
    $Name = ``*No``
Perform $Name
Cancel

*Yes
Display ``Connection established``
Return

*No
Display `` No connection established``
Return
```

\*Main is the parent routine. Both \*Yes and \*No are children of \*Main.

---

## Scoping Rules, *continued*

---

The scope of a variable is the routine in which it was created and any child of that routine. Variables created in a child routine do not exist in its parent routine.

### Example

#### PARENT.DCP

```
$date = Date ()
$time = ""
$city = "Chicago"
Perform "child1"
Perform "child2"
Display $name
Display $city
Display $street
Cancel
```

#### CHILD1.DCP

```
Display $date ;created in parent
$time = TIME () ;created in parent
$name = "Smith" ;not visible to parent
Return
```

#### CHILD2.DCP

```
Display $time ;created in parent
$city= "Dallas" ;created in parent
$street= "Clover Lane" ;not visible to parent
Return
```

The parent routine PARENT.DCP in the above example creates the variables `$date`, `$time`, and `$city` which the child routines CHILD1.DCP and CHILD2.DCP can successfully access and modify. However, since `$name` is created in CHILD1.DCP, it is known only in this script and its children. It is not known in PARENT.DCP or CHILD2.DCP. Likewise, `$street` is known only to CHILD2.DCP and its children. It is not known in PARENT.DCP or CHILD1.DCP.

An execution error occurs when the `DISPLAY $name` and `DISPLAY $street` commands are executed. An error message displays, indicating an undefined variable was referenced by the parent routine.

Subroutines within the same script file which are called by a `PERFORM` command do not follow these rules.



Also see: ARGUMENTS command

**Parameter Passing & Subroutines** section in this chapter

---

## Parameter Passing & Subroutines

---

The scoping rules defined in the previous section allow variables created in a parent routine to be passed down to any of its child routines, but do not allow a child routine to pass the value of a variable it created back to its parent. They also specify that if a child routine modifies a variable created on the parent level, it is modified on the parent level also; variables created in the parent routine have no protection from modification from within a child routine. This is the default scope of variables.

To create modular or structured scripts, it may be desirable to extend the limitations imposed by the scoping rules. DCS provides a parameter passing scheme that allows data from a parent routine to be mapped into local variables in a child routine. This mapping allows the passage of data back and forth between parent and child routines, while the creation of local variables protects variables created in the parent routine from being unintentionally modified by the child routine.

These features allow the script writer to create groups of generalized utility routines that can be called from script applications, promoting modular, structured script development.

To pass parameters, a parameter list must be specified by both the parent and child routines. A parameter list can be defined for a child routine in either of two ways. The child can specify the desired parameter list on the same line as the label indicating the start of the child routine, or in an `ARGUMENTS` command included immediately following the label.

### Example

Each of these command fragments:

```
*sub1 (%int, $string, !real, #bool)

*sub2
ARGUMENTS (%int, $string, !real, #bool)
```

defines a child routine, which specifies a parameter list containing the local variables `%int`, `$string`, `#bool`, and `!real`.

Parameters can only be passed by a parent routine using a `PERFORM` command. The parent specifies the parameters to pass by including a parameter list after the `PERFORM` command.

### Example

These two examples:

```
PERFORM sub1 ($name, %age, 25.00)
;to a label in this script file

PERFORM "Child1*sub1" ($name, %age, 25.00)
;to a label in the Child1 file
```

illustrate how to pass the parameters `$name`, `%age`, and `25.00` to the routine labeled `sub1` (either to a local subroutine or to another script file). The parent routine passes all variable parameters by reference. If the parameter is constant or an expression, it is passed by value.

When the parent passes a string, numeric, or Boolean variable to the child routine, the child routine can use the initial value of the variable and can modify the variable. The parent routine can

---

## Parameter Passing & Subroutines, *continued*

---

then use the modified value in the variable. The variable names used in the two parameter lists need not be the same, but the variables must be of the same type.

When the parent passes a string, numeric, or Boolean expression (not simple variables) to the child routine, the child routine can use or modify this value, but modifying it will not affect expressions or variables in the parent routine.

### Example

In this example:

```
PERFORM sub1 ($name, %number)
```

the variables are passed by reference; the child routine uses the variables' initial values, and can modify these variables, passing the new value back to the parent routine.

When passing parameters between two routines, there is customarily a one-to-one correspondence between parameters in the parameter list of the parent and that of the child. Corresponding parameters must be of the same type: string, numeric, or Boolean. DCS places the value of each parameter in the parent's parameter list into the corresponding variable in the child's parameter list.

### Example

In this example:

```
PERFORM sub1 ($name, %age, 25.00)
.
.
.
*sub1 ($String, %Years, !Salary)
```

three parameters are passed from the parent to the child. The value stored in \$name is passed by reference to the variable \$String. The value stored in %age is passed by reference to %Years. The value 25.00 is passed to !Salary.

Specifying a parameter list for a child routine creates local variables for the child routine. A local variable in a child routine can modify a variable in a parent routine if the parent variable is passed by reference to the child routine or if the variable name is the same in a parent and a child routine. A parameter list allows you to make a well-defined interface between the parent and child routines.

Additional local variables can be created for use by the child routine by specifying an ARGUMENTS command in addition to the parameter list. These variables are not parameters; values cannot be passed to or from them. They are local variables to be used by the child routine that are disposed of when control branches back to the parent routine. Since they are local variables, they protect variables in the parent routine which bear the same name from modification.

---

## Parameter Passing & Subroutines, *continued*

---

### Example

In this example:

```
$name = "Arnold Wilson"
%age = "34"
PERFORM sub1 ($name, %age)

*sub1 ($name, %age)
ARGUMENTS ($Date, %Bracket)
```

four local variables are created: `$name`, `%age`, `$Date`, and `%Bracket` in the routine `sub1`. The variables `$name` and `%age` are parameters and therefore receive initial values from the calling routine, while the local variables `$Date` and `%Bracket` are assigned null initial values.

---

## Tables, Records and Data Manipulation

---

One of the most important features of the DCS Script Language is the automation of communications sessions. Tables facilitate the automation process by providing a structured mechanism through which incoming and outgoing data can be processed.

Tables are temporary structures used to implement random access file operations. Tables exist in memory, and are therefore destroyed when script execution terminates. You can, however, save the contents of a table to a file at any time for permanent storage. Up to 16 tables can be created, each identified by a number from 0 through 15.

A table consists of one or more records in which the desired data is maintained. Tables contain either text records or structured records, but not a mixture of both. When a table is defined, you must specify whether the table will have text records or structured records.

### Structured Tables

A structured table is most often used to maintain data of a uniform format which consists of several organized components. These components may be of different data types (character, integer, etc.) and can be accessed as a group or individually. Each group of components is referred to as a record; each component is referred to as a field within the record. To illustrate, consider a phone book as a structured table:

Smith, Arnold	1232 Main Street	555-2233
Thomas, Cindy	4435 South Lake Street	555-7899
Zeller, Henry	7708 Maple Avenue	555-6521

Each row comprises one record. Each record consists of three fields (name, address, and phone number).

A structured table consists of fixed-length records made up of a specified number of fields. A structured table is defined with the following command:

```
TABLE DEFINE Table FIELDS f1 ... fi FILE
```

The `Table` argument specifies the table identifier (from 0 to 15). The `FIELDS` clause indicates a structured table. Text tables do not have fields. The `f1` through `fi` arguments specify the desired fields and are defined according to type and length. Each field argument includes:

- ▼ A type keyword; either CHAR, INT, or REAL (specifying character, integer number, or real number respectively), and
- ▼ An integer from 1 to 254 that specifies the length of each field.

The data in a table is stored as a string. The logical data type is retained to facilitate the import and export of data. The `TABLE DEFINE` command determines the number of fields in a record. The optional `FILE` keyword is used for structured tables only. By default, a structured table is maintained in global memory in its entirety. The `FILE` keyword stores the specified table on disk in a temporary file and swaps records in and out of global memory as needed. Use this option when working with tables larger than 64K to avoid running out of memory.



---

## Tables, Records and Data Manipulation, *continued*

---

### Example

A table used to hold the Phone Book data shown on the previous page could be defined in the following way:

```
TABLE DEFINE 1 FIELDS CHAR 20 CHAR 50 CHAR 8
```

This command defines Table 1 as a structured table with three fields. The first field holds the person's name. If any records contain names longer than 20 characters, all letters past the 20th will be truncated. The second field holds the person's address. Again, all characters past the 50th character are truncated. The third field holds the phone number. This field was designated as eight characters long to hold seven numbers and a hyphen.

A table to hold the Phone Book entry data could be defined in a number of different ways. The name could be split into two separate fields for first and last name. The phone number could drop the hyphen and be stored in two separate integer fields. When deciding how to define the fields of a structured table, carefully consider how the data will be used. The purpose of defining a table is to facilitate data manipulation.

Two types of operations can be performed on a structured table: table operations and record operations.

### Table Operations

Table operations treat the table as a unified entity. They to operate on the table as a whole, but do not access individual records. The following table operations can be used on structured tables:

Command	Description
TABLE CLEAR	Clears all data from the table.
TABLE CLOSE	Removes the specified table from memory.
TABLE COPY	Copies the contents of one table to another.
TABLE DEFINE	Creates a table.
TABLE LOAD	Imports a set of data records from a file into a table.
TABLE SAVE	Exports the entire contents of a table to a file.
TABLE SORT	Performs a table sort based on specified criteria.



Also see: **Chapter 3 Commands**

---

## Tables, Records and Data Manipulation, *continued*

---

### Record Operations

Record operations treat each record as a separate entity, and allow you to operate upon each record individually.

Data contained in a table cannot be accessed directly, but must be accessed one record at a time through the record buffer. A record buffer exists for each table defined, and is an exact template of the records in the table, reflecting the fields created. DCS accesses the record buffer by using a record buffer variable. A structured record buffer variable observes the following syntax:

```
@RTable.Field;which table = [0...15], which field = [1...n]
```

The Table argument specifies the number of the table. The optional **Field** argument is preceded by a period and specifies a field within the record buffer variable, allowing access to a single field. For example:

@R3 defines a record buffer variable for the entire record buffer associated with table three;

@R3.5 defines a record buffer variable for the fifth field of the record buffer associated with Table 3.

DCS allows you to perform the following record operations:

Record Commands	Action
RECORD READ	Reads one record of data from the specified table.
RECORD WRITE	Writes one record of data to the specified table.

RECORD FORMAT and RECORD SCAN commands are also available, but they are advanced shortcuts built upon the basic reading and writing operations. For details, see **Chapter 3 Commands**.

The RECORD READ operation reads one record from the specified table into the corresponding record buffer. The record at which to begin reading can be specified with the AT clause, allowing random access. If the AT clause is not included, DCS begins reading at the first record in the table. The first record in a table is number 0 (zero). All successive record reads are then performed sequentially. The data read can then be accessed through the record buffer variable.

#### Example 1

The following commands:

```
Record Read 6 AT 3          ;read at record 3
While Not EOF ( )         ;while not at end of file
Begin
  Display @R6              ;display record contents
  Record Read 6            ;read next record
End
```

establish a loop which displays the records in Table 6, beginning with record four (specified with the number three since the first record is numbered zero) and continuing through to the end of the file, in your terminal window (assuming table six has been defined).

---

## Tables, Records and Data Manipulation, *continued*

---

The RECORD WRITE operation writes one record from the record buffer to the corresponding table. The record at which to begin writing can be specified with the AT clause to allow random access.

If the AT clause is not included, DCS begins writing at the last record in the table. All successive record writes are then performed sequentially. The data is passed to the table through the corresponding record buffer variable.

### Example

The following commands:

```
While Not EOF ()
Begin
  Record Read 1
  If @R1.3 = ""
    @R1.3 = Time (),
  Record Write 1
End
```

perform both reads and writes to table one. Each record is read from the table and examined. If the third field of the record is empty, the current time is written to that field. This is sequentially performed on all records until the end of the file is reached.

RECORD WRITE operations write data from the record buffer to the table structure, not to your disk. If you want to permanently save data written to a table, you must execute a TABLE SAVE command to save the contents of the table to a file.

### Text Tables

A text table is most often used to maintain unstructured data which consists of sequential lines of text. Unlike structured tables, text tables do not contain fields of multiple data types. To illustrate, consider the following unformatted text as a text table:

```
His name is Arnold Smith.
He lives at 1232 Main Street.
His phone number is 555-2233.
```

Each row comprises one record; there are no fields. A text table must be used to store data of an unknown or variable length.

A text table consists of variable-length records up to 254 bytes long, where each character comprises one byte. Records in a text table are delimited by carriage return and line feed characters. A text table is defined with the following syntax:

```
TABLE DEFINE Table TEXT FileName
```

The **Table** argument specifies the table identifier (from 0 to 15).

The **TEXT** keyword indicates to DCS that a text table is being defined. Text tables do not have fields. Unlike structured tables, text tables are always maintained on disk. The **FileName** argu-

---

## Tables, Records and Data Manipulation, *continued*

---

ment specifies the name of the file upon which table operations will be performed. If the specified file does not exist, the TABLE DEFINE command creates the file.

### Example

A table used to hold the unformatted text above could be defined in the following way:

```
TABLE DEFINE 1 TEXT "MYDATA.TXT"
```

This command defines table one as a text table. All table operations will be performed upon the file MYDATA.TXT.

Since a text table consists of lines of text delimited by carriage return and line feed characters, the table definition does not allow the type of flexibility associated with a structured table.

### Table Operations

Table operations treat the table as a unified entity. The following table operations can be performed on text tables:

Command	Action
TABLE CLEAR	Clears all data from the table
TABLE CLOSE	Saves the table's contents to disk and removes the specified table from memory
TABLE COPY	Copies the contents of one table to another
TABLE DEFINE	Creates a table

### Record Operations

Record operations operate upon individual records. Record operations are performed on text tables using the record buffer variable. A text record buffer variable observes the following syntax:

```
@RTable
```

The `Table` argument specifies the number of the table. Since there are no field designations allowed in a text table, the `Field` argument is not included. For example, `@R3` defines a record buffer variable for the record buffer associated with table three.

DCS allows you to perform the following record operations:

Command	Action
RECORD READ	Reads one record of data from the specified table
RECORD WRITE	Writes one record of data to the specified table

The RECORD READ command copies one record, or depending on the position of the read pointer, the rest of a record into the corresponding record buffer. The AT clause specifies the byte position of the read pointer in a record in the table and allows the command to start copying from any position in the table. Without the AT clause, the characters are copied from the end of the previous record in the table to the end of the record where the pointer presently is, but with

---

## Tables, Records and Data Manipulation, *continued*

---

the AT clause the characters are copied from the position in a record indicated by the AT clause to the end of the record.

The position in a text table is specified as a byte number at which to begin reading in a table, not a record number as in structured tables. The first position in a table is considered position 0 (zero). DCS includes the carriage return and line feed characters that end a record when it figures the number of bytes in a table or a record. For example, if a text table were to consist of one record and if you were to assign ten bytes to that record, DCS considers the table to have 12 characters or byte positions in the table (from 0 to 11).

This text record:

```
His name is Arnold Wilson.
```

is 28 bytes long (26 characters and spaces plus one carriage return and one line feed). If this is the first record, to begin reading at the “A” in Arnold, specify `RECORD READ 0 AT 12`.

Since the records in a text table can vary in length, you can also specify the number of bytes to read with the `LENGTH` clause. If a `LENGTH` clause is not included, DCS reads until it encounters a carriage return character and then a line feed character.

If the record below:

```
His name is Arnold Wilson.
```

is the first record in Table 5 (five), to read *only the name* from this record you would specify:

```
RECORD READ 5 AT 12 LENGTH 13
```

DCS does not prevent you from reading multiple text lines if the length you specify encompasses carriage returns and line feeds. Since the maximum length of a record buffer variable is 254 characters, you may not specify a length larger than 254. When calculating the read length, remember that the carriage returns and line feeds that appear at the end of a line count as one byte each. The data read can then be accessed through the corresponding record buffer variable.

The following example:

```
RECORD READ 6 AT 0           ;read first record
WHILE NOT EOF ()           ;while not at end of file
BEGIN
  DISPLAY @R6               ;display record contents
  RECORD READ 6             ;read next record
END
```

establishes a loop which displays all the records in Table 6 in your terminal window (assuming Table 6 has been defined).

After a record read is performed, the read pointer is positioned at the character following the last character read. If the `LENGTH` clause is not included, this position is, by default, at the beginning of the next line of text.

---

## Tables, Records and Data Manipulation, *continued*

---

The RECORD WRITE operation writes one record from the record buffer to the specified table. The position at which to begin writing can be specified with the AT clause to allow random access. This position is specified as the byte (not line number) at which to begin writing, where the first byte is considered byte 0 (zero). Carriage returns and line feeds count as one byte each. If the AT clause is not included, DCS begins writing at the end of the table.

Since the records in a text table can vary in length, you can also specify the number of bytes for DCS to write with the LENGTH clause. If not included, DCS places a carriage return character and a line feed character at the end of the text written. If you specify a write position that is not at the end of the table, DCS writes over all text that is currently in the specified position (it does not “insert”).

DCS does not prevent you from writing multiple text lines if the length you specify encompasses carriage returns and line feeds. Since the maximum length of a record buffer variable is 254 characters, you cannot specify a length larger than 254. When calculating the write length, remember that the carriage returns and line feeds that appear at the end of a line count as one byte each.

### Example

The following example:

```
While True
Begin
    Collect @R6
    Record Write 6
    Increment %Count
End
```

establishes a loop which continually collects all lines of data that come into your terminal window and writes them, line by line, to Table 6 (assuming Table 6 has been defined).

After a record write is performed, the write pointer is positioned at the character following the last character written. If the LENGTH clause is not included, this position is, by default, at the beginning of the next line of text.

Tables are the mechanism through which DCS accesses file objects. This section has provided you with an overview of their purpose and use. For a more detailed explanation of the commands used to create and manipulate data stored in a table, see the TABLE and RECORD commands in **Chapter 3 Commands**.

---

## Menus

---

DCS has four default menu sets (MAIN, MEMO, SCRIPT, and SESSION) which are displayed when a DCS document (window) of the corresponding type is active. Furthermore, the four menus are associated at the DCS application level, not at the session level. This means, for example, the SESSION menu set will be used when any session window is active. The MAIN menu set is displayed if there are no open DCS child windows.

These menus may be modified (customized) through the DCS MENU EDITOR feature or through the script MENU commands. Changes made through the MENU EDITOR are permanent and become the default definition of that menu for all windows of that type. The MENU EDITOR does allow the user to reset the menu to DCS defaults. Changes made through a script are temporary and menus revert back to their default definition when the script exits or executes a MENU CANCEL command.

DCS is a Windows MDI application (see programming considerations below) allowing multiple sessions and scripts to be executing at the same time. Because the menus are considered application-level resources and most scripts operate at the session level, the following rules are applied when the MENU commands are used:

1. The first script to execute a MENU command controls the DCS MENU definition until the script exits or performs a MENU CANCEL command. The assumption is that the script wants to have total control of what the user has access to during the duration of the script. If the script executes another MENU command without canceling the previous MENU command, DCS issues a MENU CANCEL command to automatically terminate the previous menu definition.
2. The defined MENU becomes the one and only menu regardless of document type until the script exits or performs a MENU CANCEL command.
3. Any script that executes a MENU command while there is a script defined menu in place causes a run-time error to be returned and the command will not be executed. It is important that the script writer check for this error condition before proceeding with the script execution. See the MENU command for details.
4. All script menu commands, except the MENU command, are valid only after a MENU command has been performed otherwise they will cause a run-time error to be returned and the command will not be executed. It is important that the scriptwriter check for this error condition before proceeding with the script execution. See the MENU command for details.
5. All script menu functions are valid with any menu. This is because they do not modify the menu but only return the status of the menu item.

There will be times that the scriptwriter will want to use the default menu definitions or combination of definitions as the starting definition. DCS allows for this through the use of SYSTEM menu numbers. These SYSTEM numbers relate the DCS default definitions for easy identification for user selection. If the script writer uses one of the SYSTEM numbers it is related to the default DCS definitions and not to any modifications made through the MENU EDITOR feature.

---

## Menus, *continued*

---

The following SYSTEM numbers are defined:

System	Menu	Active window type*
SYSTEM 1	File	MAIN
SYSTEM 2	File	SESSION, SCRIPT, MEMO
SYSTEM 3	Edit	MEMO, SCRIPT
SYSTEM 4	Edit	SESSION
SYSTEM 5	Script	MAIN, MEMO, SCRIPT, SESSION
SYSTEM 6	View	MAIN, MEMO, SCRIPT, SESSION
SYSTEM 7	Tools	MAIN, SESSION
SYSTEM 8	Tools	MEMO, SCRIPT
SYSTEM 9	Help	MAIN, MEMO, SCRIPT, SESSION
SYSTEM 10	Window	MEMO, SCRIPT, SESSION
SYSTEM 11	Session	SESSION
SYSTEM 12	Insert	MEMO

\*The visible menus vary depending on which type of window is active (in focus). MAIN refers to the DCS application window. SESSION refers to session windows, MEMO to memo windows, and SCRIPT to a script editor window.

### Example 1

In this example:

```
MENU
  POPUP "File" SYSTEM 2
  POPUP "Edit" SYSTEM 3
  POPUP "Session" SYSTEM 11
  POPUP "Windows" SYSTEM 10
  POPUP "Help" SYSTEM 9
MENU END
PERFORM setup

MENU CANCEL
PERFORM receive_files
```

the menu bar definition is made up of the **File** menu of the MEMO/SCRIPT/SESSION group, the **Edit** menu of the MEMO/SCRIPT group, the "Session menu of the SESSION group, and the **Help** menu of the MAIN/MEMO/SCRIPT/SESSION groups. The MENU CANCEL command restores these options before the `receive_files` routine is performed.



---

## Menus, *continued*

---

### Example 2

This example:

```
MENU
  POPUP "File" SYSTEM 2
  POPUP "Edit" SYSTEM 3
  POPUP "Session" SYSTEM 11
SEPARATOR
ITEM "NewItem" PERFORM new_item_routine
  POPUP "Windows" SYSTEM 10
  POPUP "MyMenu"
ITEM "MyItem" PERFORM my_routine
  POPUP "Help" SYSTEM 9
MENU END
PERFORM setup

MENU CANCEL
  PERFORM receive_files
```

is the same as Example 1 with a new item added to the **Session** menu and a new popup menu and item inserted between the **Windows** and **Help** menus.

---

## DCS Windows & Window Handles

---

### DCS Windows

An active window is the window which has focus. It appears with a highlighted (as opposed to dimmed) title bar and window border. The active window almost always lays on top of any other open windows.

- ▼ An active session is understood to be a session currently in contact with a remote system.
- ▼ An active session window is a session which is both in contact with a remote host and also in focus.

If a window is hidden, it is not visible. However, a hidden session window can still contain an active session, and commands may be sent to the active session.

### DCS Window Handles

Window Handles specify a particular window as the target of a function or command. The target window is determined by using the:

- 1) `WinHandle` argument of the function or command
- 2) Script's `DefaultSessionHandle`
- 3) Active window

In many functions and commands, you have the option of specifying a particular window (session window, DCS child window, edit window, etc.) as the target of a function or command. To specify a particular window, its window handle must be included in the command syntax. A window handle is an integer, and in most cases is specified in a function or command by the `WinHandle` argument.

Each script has a `DEFAULTSESSIONHANDLE` variable. This variable is set to the window handle of the active window when the script is executed. If there is no DCS child window active when the script is executed the `DEFAULTSESSIONHANDLE` variable is NOT set. The `SET DEFAULTSESSIONHANDLE` command can be used to change/set this variable for the script. When set all functions or commands effect the specified window unless a `WinHandle` argument is used with the command or function. The `DEFAULTSESSIONHANDLE` function can be used to determine the current variable setting.

If you do not know the window handle of a particular window, you can retrieve the window handle using the `HWNDLIST` or `WINDOWHND` functions, or the `CONNECT` command.

#### Example

In this example:

```
%hnd = WINDOWHND ("Session1")
WINDOW HIDE %hnd
```

the `WINDOWHND` function is used to retrieve the window handle for the window titled "Session1". The handle is stored in the variable `%hnd` and used with the `WINDOW HIDE` command to hide the "Session1" window.

---

## DCS Windows & Window Handles, *continued*

---

### Programming Considerations

DCS supports the Windows MDI interface. This allows an arbitrary number of emulations, connectors, and editing windows to run concurrently. An arbitrary number of instances and sessions is also possible, limited only by the user's machine and system capabilities. Depending upon the user's needs, the environment could be:

- ▼ One instance of DCS running one session.
- ▼ One instance of DCS running several sessions concurrently.
- ▼ An arbitrary number of instances of DCS, each running a single session.
- ▼ An arbitrary number of instances of DCS, each running several sessions concurrently.

Before writing scripts which take advantage of the Windows MDI interface, you should understand how DCS scripts operate in this environment.

When a script is executed, its functions and commands affect the session window active at the time the script is executed by default (sets the `DEFAULTSESSIONHANDLE`). For example, if Window "A" is active when the script is executed, any command or function which may specify a particular window will always affect Window "A" (unless the command or function specifies a different window).

If no session window is active when the script is executed and the script does not set the `DEFAULTSESSIONHANDLE`, the script has no default window. Instead, each command and function affects, by default, whichever window is active at the time the command or function is executed (unless the command or function specifies a different window).

More than one script may be active at the same time, and in this case each script may have a different default window or the same default window. A script may set, or reset, its default window with the `SET DEFAULTSESSIONHANDLE` command. The `DEFAULTSESSIONHANDLE` function returns the current default window.

Scripts may execute other scripts, and assign a default window to the new script, with the `SPAWN` command.

---

## Event Handling - WAIT and WHEN Commands

---

Two types of commands are available for detecting events:

▼ **WAIT** commands

**WAIT** commands pause script execution until a specified event becomes **TRUE**. This is known as a wait state.

▼ **WHEN** commands

**WHEN** commands branch script execution to an indicated command block when a specified condition become **TRUE**. **WHEN** commands executed prior to the current wait state will interrupt the wait state if their conditions are met. This capability allows a script to wait for multiple events.

Both **WAIT** and **WHEN** commands are followed by an event clause and a command list. DCS executes the commands in the command list when conditions in the event clause are met.

**WAIT** commands are activated when the event clause becomes **TRUE**. **WHEN** Commands are only activated during a wait state—if the event clause becomes **TRUE** before or during the current wait state.

To terminate the current wait state, include a **RESUME** command in the **WHEN** command list. This prevents the interrupted **WAIT** from regaining control when the command list is executed.

### Example

Imagine you are connected to a system expecting a password to be sent before access is approved. When the password is sent, the system sends one of three responses:

Password Response
password ok
password invalid
old password, please update

To accommodate this system, you could use the following script:

```
WHEN STRING 0 "password ok" PERFORM get_info, RESUME
WHEN STRING 1 "password invalid" CANCEL
WHEN STRING 2 "old password" PERFORM up_passwd, RESUME
WHEN TIMER "30" PERFORM time_out, RESUME
WAIT RESUME
CANCEL
```

Four **WHEN** criteria are first established, and then a **WAIT RESUME** command executes. There are many types of **WAIT** commands, but the **WAIT RESUME** could be considered a “generic” **WAIT**. It instructs DCS to wait until it receives the instruction to resume execution. No other script execution takes place during the **WAIT**. If the event clause of any **WHEN** command becomes true while DCS is waiting, the **WHEN** is activated, and the commands in the command list are executed.

---

## Event Handling - WAIT and WHEN Commands, *continued*

---

In the example on the previous page, DCS waits 30 seconds for a response from the remote system. If “password ok” is received, execution branches to the routine labeled `get_info`. When that routine is finished, control returns to the `WAIT` statement. If “password invalid” is received, execution terminates. If “old password” is received, execution branches to the routine labeled `update_password`. If none of these three responses is received within 30 seconds, execution branches to the routine labeled `time_out`.

---

## Dynamic Data Exchange - DDE

---

Dynamic Data Exchange (DDE) allows the exchange of information between two independent Windows applications. DCS supports DDE with Script Language commands. For a complete listing of the supported DDE commands, see **Chapter 3 Commands**.

### DDE Concepts

There are two participants in any DDE conversation: a client and a server. The client is the application that initiates a conversation and makes requests. The server is the application that answers the client and fills its requests. Both the client application and the server application must be running on your computer at the same time.

There are three stages in a DDE conversation:

▼ Initiation

The client requests a conversation with the server by sending a Windows initiate message. The server responds by informing the client whether or not the requested conversation could be established.

▼ Transaction

The server responds to requests from the client. The client can send four types of requests:

Message Type	Action
Request message	Requests a specified data item from the server
Advise message	Requests continuous updates from the server on a specified data item
Poke message	Requests that the server receive the specified data item
Execute message	Requests that the server execute the specified commands

▼ Termination

At any time, either the client or the server can end the conversation by sending a Windows terminate message. The other application then answers with a terminate message.

To illustrate the advantages of DDE, consider the following example:

Imagine that you use DCS to gather weekly sales data from a remote computer system. You read the data into a table, save it in a file, and get a printout of the data. You then run a spreadsheet application using the data to update the appropriate cells on the spreadsheet. When this is completed, you run a graphics package to chart the data. To accomplish this you have taken the time to sequentially run three applications, print out the data, and enter data into two applications.

This procedure could be greatly simplified using DDE. You first run both the spreadsheet and graphics applications (memory permitting). You then gather the data as usual, and store it in a table. Since it will be sent directly to the applications that need it, there is no need to save it to a file or print it out.

---

## Dynamic Data Exchange - DDE, *continued*

---

You then execute a DDE script. This script initiates a conversation with the spreadsheet, sending the required data to the appropriate cells. It then terminates this conversation and initiates a conversation with the graphics package, sending it the required data. It then terminates this conversation and cancels script execution.

By running a single script, you have automatically updated each application with just the data it needed, without ever exiting DCS. The script may be modified to continuously send updated data to each application as it is received from the remote system. DCS can also send commands to the other applications instructing them to draw a chart or print a spreadsheet.

DDE is a quick and easy way for multiple applications to share data. DDE also allows the exchange of control between applications. DCS can only conduct a DDE conversation with another Windows application that supports DDE.

### **DDE Implementation**

In DCS, all DDE functions are performed using script commands. If DCS is to be the client, a client script must be written containing all the DDE requests DCS will make. If DCS is to be the server, a server script must be written which prepares DCS to handle incoming requests. To conduct a DDE conversation, both applications in the conversation need to agree upon the following elements:

- ▼ Server Name

A DDE conversation begins when the client establishes a conversation with a server. The client must, therefore, know the name to which the server will respond. If another Windows application is the client, DCS responds to all initiate requests using DCSSERIES as the server name. If DCS is the client, you must establish, from an application's documentation, the server name to which it will respond. Most applications use some form of their application name as their server name. For example, Microsoft Excel responds to the server name EXCEL.

- ▼ Topic

A DDE conversation must also have a topic. The topic describes something in the server application that the client wants to access. If another Windows application is the client, you must specify one of the three following DCS topics: the name of your DCS server script, a null topic, or SYSTEM.

If a server script is specified as the topic (the file extension need not be included), the establishment of the conversation depends on current DCS state. If no script is running, the specified server script is executed and the conversation begins. If another script is running and execution is paused at a `WAIT RESUME` command, the current script is closed, and the specified server script is executed. If another script is running and execution is not at a `WAIT RESUME` command, the initiation fails.

If the SYSTEM topic is specified, no script need be running for DCS to answer the initiation. However, if no script is running, DCS will be unable to process any client

---

## Dynamic Data Exchange - DDE, *continued*

---

requests. For this reason, the SYSTEM topic should only be used to test for the existence of DCS.

If DCS is the client, you must establish the topic names required by the other application. For example, Microsoft Excel recognizes the name of any open document (with the extension), or the SYSTEM topic.

### ▼ Item Name

Each DDE request must reference an item. The main purpose of this item name is merely to match a client request to the proper server response. The server dictates the format of an item name, but both client request and server response must reference the same item name. If another Windows application is the client, DCS responds to any string item name that is established in a DCS server command. If DCS is the client, you must establish the type of item names the other application requires. For example, Microsoft Excel can use a row and column reference as an item.

### Multiple DDE Channels

DCS can process up to 16 DDE conversations at the same time. The conversations can involve any number of applications (memory permitting). DCS can be client or server to any of these applications. If multiple DDE conversations are taking place, DDE requests must be directed not only to the proper application, but also through the proper DDE channel. All applications that allow multiple DDE channels have a mechanism through which the channel number is obtained during the initiate process. In DCS, a `ChannelVar` argument is included in the syntax of the ACCESS and WHEN INITIATE commands. This variable receives a channel number when the DDE conversation is established. If multiple DDE conversations are taking place, all DDE requests should reference this channel number to insure that they are directed to the desired DDE conversation.

### Single/Multiple Data Items

When acting as a DDE client, DCS can send or receive data in either of two ways: by processing a single data item, or by processing an entire table containing multiple data items. DCS can poke data to the server using the POKE command to send a single data item, or using the TABLE SEND command to send the entire contents of the specified table. DCS can request data from the server using the REQUEST command to request a single data item, or using the TABLE REQUEST command to request multiple items to be stored in a table.



---

## Dynamic Data Exchange - DDE, *continued*

---

### Syntax

The syntax of the DDE commands supported by DCS is summarized below.

DDE Message	DCS as Client	DCS as Server	Response
ADVISE	WHEN ADVISE	TABLE REPLY	
EXECUTE	INSTRUCT	WHEN EXECUTE	
INITIATE	ACCESS	WHEN INITIATE	
POKE	TABLE SEND	WHEN POKE	
REQUEST	TABLE REQUEST	WHEN REQUEST	TABLE REPLY
TERMINATE	ACCESS CANCEL	WHEN TERMINATE	

The commands in the DCS as Client column send the DDE message to the server application. The commands in the DCS as Server column prepare DCS to receive the DDE message from a client application. The Response column indicates how DCS replies to the client message.

### Example

In this example:

```
ACCESS 'Excel' 'System' %ch_num
IF ERROR () DISPLAY 'No access', CANCEL
INSTRUCT '[OPEN("C:\\EXCEL\\SURVEY.XLS")]'
```

a DDE conversation is established with Microsoft Excel, using the SYSTEM topic. The channel number associated with this DDE conversation is stored in the variable %ch\_num. If the conversation cannot be established, an error message is displayed in the terminal window, and execution terminates.

If it is established, the INSTRUCT command directs Excel to open the C:\EXCEL\SURVEY.XLS worksheet. Notice that the INSTRUCT command syntax is in the format expected by Excel.

---

## Task Errors

---

Task errors occur when DCS cannot carry out the tasks as they are presented. There are four types of task errors:

▼ Fatal Errors

When DCS encounters a fatal error, the **Fatal Task Error** dialog is displayed along with a brief explanation of the error. A fatal error indicates that DCS is not able to proceed in execution from the command in which the error was encountered. For example, executing a **PERFORM** command with a target script that does not exist generates a fatal error.

When a fatal error is encountered, DCS allows execution termination only. To terminate execution, click the **OK** button.

▼ Critical Errors

When DCS encounters a critical error, the **Task Error** dialog is displayed along with a brief explanation of the error. A critical error indicates a serious error which will likely impair the logic of the rest of the script. DCS is able to proceed in execution from the command in which the error was encountered, if instructed to do so. For example, running out of global memory would generate a critical error.

When a critical error is encountered, DCS allows either continuation or termination of execution. To continue execution, click the **OK** button. To terminate execution, click the **Cancel** button.

▼ Warning Errors

When DCS encounters a warning error, a different type of **Task Error** dialog is displayed along with a brief explanation of the error. A warning error indicates a less serious programming error. DCS is able to proceed with execution of the command in which the error was encountered, if instructed to do so. For example, violating scoping rules or attempting to read from an undefined table would generate warning errors.

When a warning error is encountered, DCS will allow either continuation or termination of execution. To continue execution, click the **OK** button. To terminate execution, click the **Cancel** button.

▼ Run-Time Errors

When DCS encounters a run-time error, no dialog is displayed. Run-time errors are not typically programming errors. They are meant to be used by a script to set status flags. For example, trying to position the read pointer to a position past the end of a table generates a run-time error. DCS continues execution past run-time errors, without informing you of their occurrence. The **ERROR** function can be used to test for the occurrence of a run-time error.

---

## Task Errors, *continued*

---

### Example

In this example:

```
TABLE DEFINE 0 CHAR 10 CHAR 15 INT 8 FILE
IF ERROR () DISPLAY 'Could not create table^M', CANCEL
TABLE LOAD 0 'DATA.DCM' AS TEXT
IF ERROR () DISPLAY 'Could not load data^M', CANCEL
RECORD READ 0
```

since each successive command is dependent on the successful execution of the previous command, the IF commands are used to test for the occurrence of run time errors.



**Note:** Using the TASKERROR command with levels and codes, displays slightly different dialogs than those described above.

---

## Converting Scripts from Previous Versions of DCS

---

Scripts written with DCS 7.0 or DCS 8.0 (all versions) are compatible with DCS 9. Scripts written with a version of the DCS Script Language prior to DCS 7.0 (with such FutureSoft products as DCS/Elite or DCS Asynchronous) must be reviewed for compatibility with the DCS Connectivity Series (DCS). In general, the majority of the scripting language syntax has not changed. However, there are key areas that have changed:

- ▼ Addressing Multiple Sessions

A script can address and affect multiple sessions. To distinguish between and among sessions, many commands and functions may use a Window Handle argument. Be sure to read the DCS Windows & Window Handles section before writing scripts which affect multiple concurrent sessions.

- ▼ Multiple, Simultaneous Scripts

You can run multiple scripts simultaneously within the DCS Connectivity Series (DCS). Event timing must be examined carefully, especially if multiple scripts are executing while multiple session windows are open. Be sure to read the Programming Considerations section before writing scripts designed to operate simultaneously.

- ▼ Configuring Sessions

Previous versions of the scripting language used the `SETTINGS` function and the `SET` commands to configure a session. These commands are still available for backward compatibility, but new features and functionality in all emulations, connectors, and file transfers are available only with the new configuration commands and configuration functions.

Specifying an emulation, connector, or file transfer is still done via the `SET EMULATION`, `SET CONNECTION`, and `SET BINARYTRANSFERS` commands, but some of the keywords may have changed. Once these have been set, you may specify configuration options appropriate to the emulation, connector, or file transfer protocol. In most cases, any feature which may be selected in the **Session Properties** dialog may also be configured via script.

For emulation configuration options, see the `EMULCONFIG` command and `GETEMULCONFIG` function.

For connector configuration options, see the `CONNCONFIG` command and `GETCONNCONFIG` function.

For file transfer configuration options, see the `XFERCONFIG` command and `GETXFERCONFIG` function.

For general session configuration option, see the `GENERALCONFIG` command and `GETGENERALCONFIG` function.

For session display options, see the `DISPLAYCONFIG` command and the `GETDISPLAYCONFIG` function.

---

## Converting Scripts from Previous Versions of DCS, *continued*

---

- ▼ Application, Menu and Toolbar Configuration

Application-level configuration options may be configured using the APPCONFIG command and GETAPPCONFIG function.

In addition, menus and toolbars can be configured via script. See **Menus** in this chapter, **Menu Functions** in Chapter 2 and **Menu and Toolbar Commands** in Chapter 3 for details.



Also see: **Appendix B Additions and Deletions**



# 2

---

## *Functions*

DCS

---

## Functions in Alphabetical Order

---



**Note:** Commands or functions flagged with an asterisk do not apply to the IBM TN3270 emulation.

### —A—

ACTIVE	Returns the handle number of the active window
ATTRIBUTES	Returns the file attributes of the specified file

### —B—

BAND	Returns the bitwise AND value of its arguments
BNOT	Returns the bitwise complement of its argument
BOOL	Converts an integer numeric to its boolean value
BOR	Returns the bitwise OR of its arguments
BXOR	Returns the bitwise exclusive OR of its arguments
BUFFER	Returns the text from a screen buffer field.

### —C—

CHR	Converts a numeric to its ANSI character value
CONNECT	Returns the connection state with the remote system
CONNECTMESSAGE*	Returns the last dial status of a modem sent to the computer
CONNECTRESULT*	Returns the last dial status of a modem sent to DCS
CURSOR	Returns the current cursor position

### —D—

DATE	Returns a current or derived date as a string
(DDE) ADVISE	Indicates that a DDE ADVISE or UNADVISE message activated a WHEN ADVISE command
DECRYPT	Returns the decrypted value of the specified encrypted string
DEFAULTSESSIONHANDLE	Returns the handle of the script's default session window
(DIALOG) CHECKBOX	Determines if the specified check box is checked
(DIALOG) EDITTEXT	Returns the contents of the specified edit text control
(DIALOG) LISTBOX	Indicates which item in the specified list box is selected
(DIALOG) MESSAGEBOX	Displays a message box
(DIALOG) RADIOGROUP	Indicates which radio button in a radio group is selected
DIRECTORY	Returns the current working directory for the specified data file type
DISKSPACE	Returns the number of bytes free on the specified drive



---

## Functions in Alphabetical Order, *continued*

---

### —E—

ENCRYPT	Returns an encrypted equivalent of the specified string
EOF	Indicates if the end of the table has been reached
ERROR	Indicates whether an error occurred during script execution
EXISTS	Determines whether the specified file exists
EXFLDATTR	Returns the extended field attributes of the specified field

### —F—

FILESIZE	Returns the number of bytes in the specified file
FILTER	Replaces specific characters in the specified string
FLDATTR	Returns the field attributes of the specified string
FLDATTREXPOS	Returns extended field attributes of the field at the specified row and column
FLDLLEN	Returns the length of the specified field
FLDNUM	Returns the field number of the specified field
FLDPOS	Returns the absolute position of the beginning of the specified field
FLDTEXT	Returns text from a screen field.

### —G—

GETAPPCONFIG	Returns a string identifying general application options
GETCONNCONFIG	Returns a string identifying current connector setting
GETDISPLAYCONFIG	Returns the display settings of a session window
GETEMULCONFIG	Returns a string identifying current emulation setting
GETGENERALCONFIG	Returns a string identifying general session options
GETPROFILEDATA	Returns a string associated with an INI entry
GETXFERCONFIG	Returns a string identifying current file transfer setting

### —H—

HWNDLIST	Returns the handle numbers of all child windows
----------	---

### —I—

ICONIC	Determines whether a window is minimized
INT	Converts a numeric to an integer numeric

### —L—

LENGTH	Returns the number of characters in the specified string
--------	--

---

## Functions in Alphabetical Order, *continued*

---

### —M—

(MENU) CHECKED	Returns the state of a checked or unchecked menu item
(MENU) ENABLED	Returns the state of a disabled or enabled menu item

### —N—

NETID*	Returns the network ID set by the current Phone Book entry
NEXT	Returns the next file name from the list created by ROUTE
NUM	Converts a string to its numeric value

### —O—

ORD	Converts a string to its ANSI numeric value
-----	---

### —P—

PASSWORD*	Returns the password set by the current Phone Book entry
PHONENUMBER*	Returns the phone number set by the current Phone Book entry
POS	Returns the position of a given string in the specified string
POSITION	Returns the size and position of the specified window
POWER	Returns the base argument raised exponentially
PRTMETRICS	Returns the current print parameters
PUTPROFILEDATA	Creates or updates an INI entry

### —R—

RANDOM	Returns a random number
RESULT	Returns the contents of the result string
REAL	Converts a numeric to a real numeric
ROUND	Rounds a real numeric to the specified decimal places
ROUTE	Creates a list of files matching the specified file type

### —S—

SCREEN	Retrieves a line of screen data at the specified position
SEARCH	Returns the position of the specified string in a session window
SEARCHINRECT	Returns the position of the specified string in a specified rectangular screen area in a session window
SECONDS	Returns a current or derived date and time as a numeric
SETTINGS	Returns the current values of the settings options

---

## Functions in Alphabetical Order, *continued*

---

STR	Converts a numeric to a string
SUBSTR	Returns a portion of the specified string
SYSMETRICS	Returns the current system parameters
SYSTEM*	Returns information about the current environment
<b>—T—</b>	
TIME	Returns a current or derived time as a string
TIMER	Returns the elapsed time since the last timer reset
TRIM	Removes a given string from the specified string
TYPEDLIBRARYCALL	Returns a string representing a return value of a DLL function call
<b>—U—</b>	
UPPER	Converts a string to its uppercase equivalent
USERID*	Returns the user ID set by the current Phone Book entry
<b>—V—</b>	
VISIBLE	Indicates whether the specified window is hidden
VERSION	Returns the current DCS version number
<b>—W—</b>	
WINDOW	Indicates if the window is a child of the DCS window
WINDOWHND	Returns the handle of the DCS window specified by name
WINDOWNAME	Returns the full name of the DCS window specified by handle
WNDCLASS	Indicates the DCS window type of the specified window
WNDFILE	Returns the disk name of the file associated with the specified window
WNDTITLE	Returns the title of the specified window
<b>—Z—</b>	
ZOOMED	Indicates whether the specified window is maximized

---

## Functions by Category

---



**Note:** Commands or functions flagged with an asterisk do not apply to the IBM TN3270 emulation.

### Boolean Functions

BAND	Returns the bitwise AND value of its arguments
BNOT	Returns the bitwise complement of its argument
BOOL	Converts an integer numeric to its boolean value
BOR	Returns the bitwise OR of its arguments
BXOR	Returns the bitwise exclusive OR of its arguments

### Configuration Functions (Details)

GETAPPCONFIG	Returns a string identifying general application options
GETCONNCONFIG	Returns a string identifying current connector setting
GETDISPLAYCONFIG	Returns the display settings of a session window
GETEMULCONFIG	Returns a string identifying current emulation setting
GETGENERALCONFIG	Returns a string identifying general session options
GETXFERCONFIG	Returns a string identifying current file transfer setting

### Conversion Functions

BOOL	Converts an integer numeric to its boolean value
CHR	Converts a numeric to its ANSI character value
INT	Converts a numeric to an integer numeric
NUM	Converts a string to its numeric value
ORD	Converts a string to its ANSI numeric value
REAL	Converts a numeric to a real numeric
STR	Converts a numeric to a string
UPPER	Converts a string to its uppercase equivalent

### Data Searching and Capturing Functions

BUFFER	Returns the text from a screen buffer field.
SCREEN	Retrieves a line of screen data at the specified position
SEARCH	Returns the position of the specified string in a session window
SEARCHINRECT	Returns the position of the specified string in a specified rectangular screen area in a session window

---

## Functions by Category, *continued*

---

### Dialog Functions

(DIALOG)HANDLE	Returns the window handle of the specified dialog box
(DIALOG) CHECKBOX	Determines if the specified check box is checked
(DIALOG) EDITTEXT	Returns the contents of the specified edit text control
(DIALOG) LISTBOX	Indicates which item in the specified list box is selected
(DIALOG) MESSAGEBOX	Displays a message box
(DIALOG) RADIOGROUP	Indicates which radio button in a radio group is selected

### Dynamic Data Exchange Functions

(DDE) ADVISE	Indicates that a DDE ADVISE or UNADVISE message activated a WHEN ADVISE command
TYPEDLIBRARYCALL	Returns a string representing a return value of a DLL function call

### Field Functions

EXFLDATTR	Returns the extended field attributes of the specified field
FLDATTR	Returns the field attributes of the specified string
FLDATTREXPOS	Returns extended field attributes of the field at the specified row and column
FLDLEN	Returns the length of the specified field
FLDNUM	Returns the field number of the specified field
FLDPOS	Returns the absolute position of the beginning of the specified field
FLDTEXT	Returns text from a screen field.

### File Functions

ATTRIBUTES	Returns the file attributes of the specified file
DIRECTORY	Returns the current working directory for the specified data file type
DISKSPACE	Returns the number of bytes free on the specified drive
EXISTS	Determines whether the specified file exists
FILESIZE	Returns the number of bytes in the specified file
NEXT	Returns the next file name from the list created by ROUTE
ROUTE	Creates a list of files matching the specified file type
WNDFILE	Returns the disk name of the file associated with the specified window

---

## Functions by Category, *continued*

---

### Math Functions

BAND	Returns the bitwise AND value of its arguments
BNOT	Returns the bitwise complement of its argument
BOR	Returns the bitwise OR of its arguments
BXOR	Returns the bitwise exclusive OR of its arguments
POWER	Returns the base argument raised exponentially
RANDOM	Returns a random number
ROUND	Rounds a real numeric to the specified decimal places

### Menu Functions

(MENU) CHECKED	Returns the state of a checked or unchecked menu item
(MENU) ENABLED	Returns the state of a disabled or enabled menu item

### String Functions

DECRYPT	Returns the decrypted value of the specified encrypted string
ENCRYPT	Returns an encrypted equivalent of the specified string
FILTER	Replaces specific characters in the specified string
LENGTH	Returns the number of characters in the specified string
NETID*	Returns the network ID set by the current Phone Book entry
PASSWORD*	Returns the password set by the current Phone Book entry
PHONENUMBER*	Returns the phone number set by the current Phone Book entry
POS	Returns the position of a given string in the specified string
SUBSTR	Returns a portion of the specified string
TRIM	Removes a given string from the specified string
UPPER	Returns the uppercase equivalent of the specified string
USERID*	Returns the user ID set by the current Phone Book entry

### System Functions

CURSOR	Returns the current cursor position
DATE	Returns a current or derived date as a string
ERROR	Indicates whether an error occurred during script execution
GETPROFILEDATA	Returns a string associated with an INI entry
PRTMETRICS	Returns the current print parameters
PUTPROFILEDATA	Creates or updates an INI entry
RESULT	Returns the contents of the result string
SECONDS	Returns a current or derived date and time as a numeric
SETTINGS	Returns the current values of the settings options

---

## Functions by Category, *continued*

---

SYSMETRICS	Returns the current system parameters
SYSTEM*	Returns information about the current environment
TIME	Returns a current or derived time as a string
TIMER	Returns the elapsed time since the last timer reset
VERSION	Returns the current DCS version number

### Table Functions

EOF	Indicates if the end of the table has been reached
-----	--

### Telecommunications Functions

CONNECT	Returns the connection state with the remote system
CONNECTMESSAGE*	Returns the last dial status of a modem sent to the computer
CONNECTRESULT*	Returns the last dial status of a modem sent to DCS
GETCONNCONFIG	Returns a string identifying current connector setting

### Window Functions

ACTIVE	Returns the handle number of the active window
DEFAULTSESSIONHANDLE	Returns the handle of the script's default session window
HWNDLIST	Returns the handle numbers of all child windows
ICONIC	Determines whether a window is minimized
POSITION	Returns the size and position of the specified window
VISIBLE	Indicates whether the specified window is hidden
WINDOW	Indicates if the window is a child of the DCS window
WINDOWHND	Returns the handle of the DCS window specified by name
WINDOWNAME	Returns the full name of the DCS window specified by handle
WNDCLASS	Indicates the DCS window type of the specified window
WNDFILE	Returns the disk name of the file associated with the specified window
WNDTITLE	Returns the title of the specified window
ZOOMED	Indicates whether the specified window is maximized

---

## ACTIVE

---

### ACTIVE ( )

The ACTIVE function returns the window handle number of the active child window.

#### Arguments

The ACTIVE function takes no arguments.

#### Result

The result is an integer expression specifying the handle of the active child window. If no child window is open, the result is 0 (zero).

#### Comments

While the ACTIVE function returns the window handle of any DCS child window, most commands and functions require the window handle of a session window (rather than a memo or script window).

#### Example

In this example:

```
%WindowHND = ACTIVE ( )
```

the variable %WindowHND contains the window handle of the active window. The variable can then be used as a window handle parameter for any command or function that calls for a window handle.



---

# ATTRIBUTES

---

## ATTRIBUTES (FileName)

The ATTRIBUTES function returns the file attributes of the specified file.

### Arguments

#### FileName

The FileName argument is a string specifying the name of a file. The FileName argument must specify a valid file name for your system.

### Result

If the file exists, the result is an integer which is the sum of the values of its attributes. File attributes have the following numeric values:

Value	File Attribute
1	Read-Only File
2	Hidden File
4	System File
16	Subdirectory
32	Archived File
64	Compressed File

### Comments

If the specified file does not exist, the ERROR function returns TRUE and the ATTRIBUTES function returns -1 (negative one).

### Example

In this example:

```
%attrib = ATTRIBUTES ("C:\MSDOS.SYS")
```

MSDOS.SYS (located in the root directory of the C : drive) has the attributes Read-Only, Hidden File and System File. The sum of the values of these attributes is 7 (1+2+4). The ATTRIBUTES function assigns the sum to the variable %attrib.

In this example:

```
$PATH = DIRECTORY (SETTINGS)  
%attrib = ATTRIBUTES ($PATH | "fsebbs.ses")
```

the DIRECTORY function obtains the path of the default location for session files, which is then included as a part of the FileName argument for the ATTRIBUTES function. If fsebbs.ses is an archived file, %attrib assumes the value 32.

---

# BAND

---

## BAND (Source, Mask)

The BAND function performs a bitwise AND operation on the **Source** argument.

### Arguments

#### Source

The **Source** argument is an integer expression.

#### Mask

The **Mask** argument is an integer expression.

### Result

The result is an integer equivalent to the **Source** argument, but with bits turned off as specified by the **Mask** argument:

Source	Mask	Result
0	0	0
0	1	0
1	0	0
1	1	1

### Comments

Use a **Mask** argument containing a zero for each bit to turn off and a one for each bit to leave unmodified. Any representation of the number (binary, decimal, octal, or hexadecimal) can be used.

The BAND function returns a 32-bit value.

### Example

In this example:

```
%result = BAND (%source, 0xFFFFFFFFE)
```

which uses the hexadecimal representation 0xFFFFFFFFE as a mask, `%result` contains the contents of `%source`, but the least significant bit in `%result` will be 0 (zero - the bit is turned off).

---

# BNOT

---

## BNOT (Numeric)

The BNOT function performs a bitwise complement operation (binary zeroes become ones, binary ones become zeroes) on its argument.

### Arguments

#### Numeric

The Numeric argument is an integer expression.

### Result

The result is an integer that is the complement of the Numeric argument:

BNOT	Result
0	-1
1	-2

### Comments

The BNOT function returns a 32-bit value.

### Example

In this example:

```
%result = BNOT (%source)
```

the BNOT function returns the complement of the %source variable. If %source originally contains the value 0x000000FE, %result assumes the value 0xFFFFF00 after DCS performs the BNOT function.

---

# BOOL

---

## BOOL (Integer)

The BOOL function returns a Boolean value corresponding to a specified integer.

### Arguments

#### Integer

The Integer argument specifies the integer value to convert.

### Result

The result is a Boolean value corresponding to the Integer argument. If the integer equals 0 (zero), the result is FALSE; otherwise the result is TRUE.

### Example

In this example:

```
#nonzero = BOOL (%product)
```

if the integer variable %product has a value of 0 (zero), #nonzero assumes the value FALSE.

If the integer variable %product has a value other than 0 (zero), #nonzero assumes the value TRUE.

---

# BOR

---

## BOR (Source, Mask)

The BOR function performs a bitwise OR operation on the `Source` argument.

### Arguments

#### Source

The `Source` argument is an integer expression.

#### Mask

The `Mask` argument is an integer expression.

### Result

The result is an integer value equivalent to the `Source` argument, but with bits turned on as specified by the `Mask` argument:

Source	Mask	Result
0	0	0
0	1	1
1	0	1
1	1	1

### Comments

Use a `Mask` argument containing a 1 (one) for each bit to turn on and a 0 (zero) for each bit to leave unmodified. You may use any representation of the number (binary, decimal, octal, or hexadecimal).

The BOR function returns a 32-bit value.

### Example

In this example:

```
%result = BOR (%source, 0x00000001)
```

using the binary representation `0x00000001` as a mask, `%result` will contain the contents of `%source`, but the least significant bit in `%result` will be 1 (one - the bit is turned on).


---

# BUFFER

---

## **BUFFER (Row, Column, Length, WinHandle)**

The **BUFFER** function retrieves a line or portion of a line of data at the specified position in a screen buffer. This function retrieves the text even when the field is hidden.

 This function applies only to 3270 or 5250 emulations.

### **Arguments**

#### **Row**

The **ROW** argument is an integer specifying the row number of the desired data. The first row is considered row zero.

For those emulations which display data in a status line, the data can be captured from the top and bottom status lines by using the following integers in the **ROW** argument:

Status Line	Integer
Top	-1
Bottom	-2

#### **Column**

The optional **Column** argument is an integer specifying the column number of the desired data. The first column is considered column zero. If the **Column** argument is not included, a default column offset of zero is assumed.

#### **Length**

The optional **Length** argument is a numeric specifying the length (in characters) of the desired data. If the **Length** argument is not included, or if **Length** exceeds the length of the line, the remainder of the line is collected.

If you use the **Length** argument, you must also include the **Column** argument. If you do not include the **Column** argument, the script will compile, but the **BUFFER** function may not perform as expected.

`BUFFER (1, 3, 5, 0x4BD3)` (Row, Column, Length, & WinHandle arguments)

*or*

`BUFFER (1, 3)` (Row & Column arguments only)

*but not*

`BUFFER (1, , 5)` (incorrect usage)

---

## **BUFFER, *continued***

---

### **WinHandle**

The optional **WinHandle** argument is an integer specifying a particular window in DCS. The inclusion of this argument allows a script in DCS to retrieve a string from a particular session window of DCS.

### **Result**

The result is a string value containing the buffer data within the specified area.

### **Comments**

If the **WinHandle** argument is not included in this function, DCS uses the window handle of the active session window.

The **ERROR** function returns **TRUE** if the **WinHandle** argument is not valid.

### **Examples**

In this example:

```
$T = BUFFER (0)
```

the string variable `$T` assumes the string value of the first line of data in the buffer.

In this example:

```
$chars = BUFFER (1, 3, 7)
```

the string variable `$chars` assumes the string value of seven characters of buffer data, starting at the second row and fourth column.

---

## BXOR

---

### BXOR (Source, Mask)

The BXOR function performs a bitwise exclusive OR operation on the **Source** argument.

#### Arguments

##### Source

The **Source** argument is an integer expression.

##### Mask

The **Mask** argument is an integer expression.

#### Result

If both the **Source** and **Mask** arguments contain the identical bit, the result is the integer 0 (zero); if one of the arguments contains a zero bit, and the other contains a one bit, the result is the integer 1 (one):

Source	Mask	Result
0	0	0
0	1	1
1	0	1
1	1	0

#### Comments

The BXOR function returns a 32-bit value. Any representation of the number (binary, decimal, octal, or hexadecimal) can be used.

#### Example

In this example:

```
%result = BXOR (0x005, 0x00A)
```

the variable `%result` contains an integer where the four least significant bits become ones. The result contains 0x00F (hexadecimal) or 15 (decimal).



---

# CHR

---

## CHR (Numeric)

The CHR function returns the string representation of its Numeric argument.

### Arguments

#### Numeric

The Numeric argument is the value to be converted. If the value is greater than 255, DCS divides the Numeric argument by 256 and uses the remainder (the modulus) as the numeric value.

### Result

The result is the ASCII representation of the value of the Numeric argument.

### Comments



Tip: For more information on numeric representations of ASCII characters, see the “Character Sets” section in the DCS 9 Online Reference.

### Example

In this example:

```
$letter = CHR (65)
```

the string variable `$letter` assumes the value A.

In this example:

```
KEY Back CHR (127)
```

the CHR function returns the delete character (the character the [DELETE] key sends), and the KEY command replaces the character the [BACKSPACE] key sends with the delete character.

In this example:

```
SEND CHR (27) | CHR (43)
```

The first CHR function returns the Escape character, and the second CHR function returns the Plus character. The Escape and Plus characters are concatenated (joined) together into a string with the concatenation operator (`|`). The SEND command then sends the string, followed by a carriage return character, to a remote system.

---

## **CHR, *continued***

---

In this example:

```
%index = 0x41
WHILE %index <= 0x5A
  BEGIN
    DISPLAY CHR (%index)
    INCREMENT %index
  END
DISPLAY CHR (0x0D) | CHR (0x0A)
```

These commands show a loop that displays the uppercase alphabet (from hexadecimal values) in the session window, followed by a carriage return (hexadecimal 0D) and a line feed (hexadecimal 0A).

---

# CONNECT

---

## CONNECT ( )

The `CONNECT` function indicates whether a session is connected.

### Arguments

This function requires no arguments.

### Result

The result is a Boolean value, where `TRUE` indicates that the session connection is active and `FALSE` indicates that the session connection is not active.

### Comments

If the `WinHandle` argument is not included in this function, DCS uses the window handle of the active session window.

The `ERROR` function returns `TRUE` if the `WinHandle` argument is not valid.

If you are compiling a script written for a previous version of DCS, make sure all uses of the `CONNECT` function are followed by parentheses (even if empty).



**Note:** DCS waits until a connection is made before returning a value. When the ComDirect (Serial) connector is used, the result is returned almost immediately as long as a port is found. With the ComTAPI (Modem) connector, there will be a delay.



**Note:** If you are connected to a COM port, the `CONNECT` function always returns `TRUE`, whether or not the session has established a connection.

### Example

In this example:

```
IF NOT CONNECT ( )
  GOTO the_end
```

if no connection is detected, `CONNECT` returns `FALSE` and control branches to the line labeled `the_end`. By using this function to check for an active connection, the script can prevent the execution of commands which require a host connection.

---

# CONNECTMESSAGE

---

## CONNECTMESSAGE ( )

The **CONNECTMESSAGE** function returns the current value of the ConnectMessage system variable.



Note: This function does not apply to the IBM TN3270 emulation.

### Arguments

This function requires no arguments.

### Result

The result is a string representing the current value of the ConnectMessage system variable.

### Comments

Compare with the SET CONNECTMESSAGE command.

### Example

In this example:

```
IF CONNECTMESSAGE ( ) = "BUSY"  
  DISPLAY "aw shucks"
```

if the last modem dial status message is `BUSY`, DCS displays the text "aw shucks" in the active session window.

---

# CONNECTRESULT

---

## CONNECTRESULT (WinHandle)

The CONNECTRESULT function returns the current value of the ConnectResult system variable.



Note: This function does not apply to the IBM TN3270 emulation.

### Arguments

#### WinHandle

The optional **WinHandle** argument is an integer identifying a particular DCS session window. The inclusion of this argument allows a script in DCS to check the status message of a particular session.

### Result

The result is an integer representing the current value of the ConnectResult system variable.



Also see: SET CONNECTRESULT command for a list of possible result values.

### Comments

Compare with the SET CONNECTRESULT command.

### Example

In this example:

```
IF CONNECTRESULT () = 0
  DISPLAY "We are there!"
```

if the last modem dial status number is 0 (zero), DCS displays the text "We are there!" in the active session window.

---

# CURSOR

---

## CURSOR (WinHandle)

The CURSOR function returns the host cursor position in a window's work space.

### Arguments

#### WinHandle

The optional **WinHandle** argument is an integer that identifies a particular DCS child window. The inclusion of this argument allows a script in DCS to find the position of the host cursor in a particular window.

### Result

The result is an integer, calculated using the following formula:

$$(\text{row} * \text{column width}) + \text{column}$$

The first row is row 0 (zero) and the first column is column 0 (zero).

### Comments

If the **WinHandle** argument is not included in this function, DCS uses the window handle for the active child window.

The ERROR function returns TRUE if the **WinHandle** argument is not valid.

Session windows display two cursors. One cursor is the host cursor. Its position in the session window is determined by the remote system to which DCS is connected. The second cursor is the local mouse cursor. This cursor shows where your mouse is presently pointing on the Windows desktop (such as other windows or other applications). In the session window, you can use the mouse cursor to determine where you are making a selection, and you can interact with the remote system using the session's mouse settings.

### Example

In this example:

```
%pos = cursor ()
%row = %pos / 80
%column = %pos % 80
```

if the cursor is on the second line (row one) at the fifth column (column four), and the column width is 80, the variable `%pos` assumes the value 84. This value is derived using the formula above, as follows:  $(1 * 80) + 4 = 84$ . The next two lines decode the `%pos` variable into the row (`%row = 1`) and column (`%column = 4`).

---

# DATE

---

## DATE (Seconds)

The DATE function returns a date string, formatted according to the date format you have chosen via the Windows Control Panel.

### Arguments

#### Seconds

The optional **Seconds** argument is an integer specifying a number of seconds. DCS uses 12:00:00 A.M. January 1, 1904 as a base (or zero seconds), adds the **Seconds** argument, and returns a date. If you do not include the **Seconds** argument, DCS returns the current system date.

### Result

The result is a string which specifies a date and is formatted according to the format you have selected in the Windows Control Panel. For example, the date might have the following format:

month/day/year



Also see: SECONDS function

### Example

In this example:

```
$date = DATE ()
```

if the current machine date is November 3, 2001, `$date` might assume the string value 11/03/2001 (depending on your system configuration).

---

## (DDE) ADVISE

---

### ADVISE ( )

The (DDE) ADVISE function indicates whether the most recently activated (DDE) WHEN ADVISE command was activated in response to an advise message or an unadvise message.

### Arguments

The (DDE) ADVISE function takes no arguments.

### Result

The result is a Boolean value where TRUE indicates an advise message was received and FALSE indicates an unadvise message was received.

### Example

In this example:

```
WHEN ADVISE 0 "item 0"  
BEGIN  
  IF ADVISE (  
    PERFORM advisesub  
  ELSE  
    PERFORM unadvisesub  
  END  
  .  
  .  
  *advisesub  
  .  
  .  
  *unadvisesub
```

If ADVISE returns TRUE, control branches to the line labeled \*advisesub. If ADVISE returns FALSE, control branches to the line labeled \*unadvisesub.



---

# DECRYPT

---

## DECRYPT (EncryptedString, Key)

The DECRYPT function returns the original characters of a string encrypted with the ENCRYPT function.

### Arguments

#### EncryptedString

The EncryptedString argument contains the string produced by the ENCRYPT function.

#### Key

The Key argument is a string containing the same string used as a key by the ENCRYPT function to produce the string contained in EncryptedString.

### Result

The result is a string value containing the original string.

### Comments

The encryption scheme in DCS can produce an encrypted string which contains significantly more characters than the original string.

 Also see: ENCRYPT function

### Example

In this example:

```
Table Define 0 Fields Char 64 Char 48
.
.
.
@R0.1 = $Name
@R0.2 = Encrypt ($Balance, $Key)
Record Write 0
;Begins a loop to gather names and balances
Table Save 0 To "C:\Balnc\" | $filename as Text
;Saves names and balances to a file
.
.
.
```

---


## DECRYPT, *continued*

---

```
Record Read 0
;Begins a loop to display names and balances
Dialog
Message @R0.1
Message Decrypt (Trim (@R0.2,, " "), $Key)
Dialog End
Wait Delay "5"
.
.
.
```

sensitive information is saved in the second field of a structured table. The example focuses on using the contents of a field and the `DECRYPT` function and, therefore, does not contain looping structures.

The script encrypts the content of the `$Balance` variable and then places the encrypted string into the second field of a record. The number of characters allowed in the second field is about three or four times larger than the number of characters in the original string.

 **Note:** If the script does not allow enough character positions in a field for all of the characters in the encrypted string, DCS discards the characters that cannot fit into the field. If characters are discarded from, or added to, an encrypted string, DCS will not decrypt the string.

Later, the script reads a record from the table and decrypts the second field of the record. However, since the second field is larger than the original string (or than the encrypted string), the script must delete any spaces from the end of the field before the `DECRYPT` function will decrypt the encrypted string. The `TRIM` function removes the extra spaces.

---

# DEFAULTSESSIONHANDLE

---

## DEFAULTSESSIONHANDLE ()

The DEFAULTSESSIONHANDLE function returns an integer identifying the window handle of the default session window.

### Arguments

The DEFAULTSESSIONHANDLE function takes no arguments.

### Result

The DEFAULTSESSIONHANDLE function returns the integer window handle of the default session window for this script. If no default session handle is defined, a -1 (negative one) is returned.

### Comments



Also see: **Windows & Window Handles** section for more information about the default session window.

SET DEFAULTSESSIONHANDLE command

### Example

```
%SessionHandle = DefaultSessionHandle ()
```

---

## (DIALOG) CHECKBOX

---

### CHECKBOX (*ControlNum*, *DialogIndex*)

The (DIALOG) CHECKBOX function is used with the DIALOG command to indicate a selected (enabled) check box in the dialog.

#### Arguments

##### ControlNum

The optional **ControlNum** argument contains a value, from 1 to *n*, to identify a specific check box, where *n* is the number of check box controls in the dialog box. If only one check box exists in the dialog, use of the **ControlNum** argument is not required.

##### DialogIndex

The optional **DialogIndex** argument is an integer (from 0 to 15) to indicate the dialog that contains the check box to test. If the **DialogIndex** argument is not included, the default value of zero is used.

#### Result

The result is a numeric value, where 1 (one) indicates that the check box control is checked and 0 (zero) indicates unchecked. If the check box control is invalid, this function returns -1 (negative one).

#### Comments

This function returns invalid results if the DIALOG CANCEL command is executed before the (DIALOG) CHECKBOX function. That is, if the dialog box no longer exists, the result of this function is invalid.

 Also see: (DIALOG) CHECKBOX command

#### Example

These commands:

```
IF CHECKBOX (3) = 1
  PERFORM read_file
ELSE
  PERFORM write_file
```

determine which subroutine to perform. If the third check box in the dialog is checked, `read_file` is performed. If it is unchecked, `write_file` is performed.

---

## (DIALOG) EDITTEXT

---

### EDITTEXT (ControlNum, DialogIndex)

The (DIALOG) EDITTEXT function is used with the DIALOG command. It returns the current contents of the specified edit text control.

#### Arguments

##### ControlNum

The optional **ControlNum** argument is a value (from 1 to *n*) specifying an edit text control, where *n* is the number of edit text controls in the dialog box. If only one edit text control exists in the dialog, inclusion of the **ControlNum** argument is not required.

##### DialogIndex

The optional **DialogIndex** argument is an integer (from 0 to 15) that indicates the dialog that contains the edit text control to test. If the **DialogIndex** argument is not included, **DialogIndex** defaults to zero (0).

#### Result

The result is a string value containing the current contents of the specified edit text control. If the specified edit text control does not exist, the result is a null string.

#### Comments

The (DIALOG) EDITTEXT function returns a result only while the dialog box is still active (prior to a DIALOG CANCEL command). That is, if the dialog box no longer exists, the result of this function is invalid.



Also see: (DIALOG) EDITTEXT command

#### Example

In this example:

```
$answer_2 = EDITTEXT (2)
```

the string variable `$answer_2` assumes the value of the characters in the second edit text control.

---

## (DIALOG) LISTBOX

---

### LISTBOX (**ControlNum**, **DialogIndex**)

The (DIALOG) LISTBOX function is used with the DIALOG command to indicate which list box item is selected.

#### Arguments

##### ControlNum

The optional **ControlNum** argument is a value (from 1 to  $n$ ) specifying a list box, where  $n$  is the number of list boxes in the dialog box. You do not need to include the **ControlNum** argument if there is only one list box in the dialog box.

##### DialogIndex

The optional **DialogIndex** argument is an integer (from 0 to 15) indicating which dialog contains the list box to test. If you do not include the **DialogIndex** argument, DCS uses the default **DialogIndex** value of zero (0).

#### Result

The result is an integer numeric value corresponding to the record number of the selected list box item. If no item is selected, the result is -1 (negative one). If a list box control that does not exist is specified, the result is also -1 (negative one).

#### Comments

The (DIALOG) LISTBOX function returns a result only while the specified dialog box is still active (prior to the DIALOG CANCEL command). That is, if the dialog box no longer exists, the result of this function is invalid.

 Also see: (DIALOG) LISTBOX command

#### Example

In this example

```
%1 = LISTBOX (1)
IF (%1 <> -1)
RECORD READ 0 AT %1
```

if the third item in the specified list box has been selected, the numeric variable %1 assumes the value 2 (records are numbered from 0 to  $n-1$ ). DCS can then execute a RECORD READ command to retrieve the data stored in record two.

---

## (DIALOG) MESSAGEBOX

---

### MESSAGEBOX (Message, Title, Style)

The MESSAGEBOX function displays a message dialog box in the specified style with the supplied title and message and returns the integer value for the selected user action (depending on the selected style).

#### Arguments

##### Message

The **Message** argument is a string variable containing the message to be displayed in the dialog box.

##### Title

The **Title** argument is a string variable containing the title to be displayed in the title bar of the dialog box.

##### Style

The **Style** argument is an integer value that describes the style of the message dialog box. The defined values can be added together to be used in combination.

In the table below, the first six values define the text and number of buttons to display below the Message. The last four values define the ICON symbol that will be displayed to the left of the Message.

Description	Style Value
OK	0
OK - CANCEL	1
ABORT – RETRY - IGNORE	2
YES – NO - CANCEL	3
YES - NO	4
RETRY - CANCEL	5
STOP – ERROR ICON	16
QUESTION ICON	32
WARNING ICON	48
INFORMATION ICON	64

---

## (DIALOG) MESSAGEBOX, *continued*

---

### Results

The result is an integer value corresponding to the style button selected by the user. This value is used to determine the action to be taken.

### Return values

Description	Return Value
OK	1
CANCEL	2
ABORT	3
RETRY	4
IGNORE	5
YES	6
NO	7

### Comments

The ERROR function is set to TRUE if the message box can not be displayed for any reason.

### Examples

This example:

```
%ret = messagebox ("Test Message", "3 Button Box", 3)
If %ret = 6 GOTO YES_Process
If %ret = 7 GOTO NO_Process
If %ret = 2 GOTO CANCEL_Process
```

displays a Message dialog with the Title set to "3 Button Box", the String "Test Message" as the Message text, three buttons below the message labeled YES, NO and CANCEL, and no ICON is displayed.

This example:

```
%ret = messagebox ("Test Message 2", "2 Button Box with
ICON", 17)
If %ret = 1 GOTO OK_Process
If %ret = 2 GOTO CANCEL_Process
```

displays a message dialog with the Title set to "2 Button Boxwith ICON", the String "Test Message 2" as the message text, two buttons below the message labeled OK and CANCEL, and the STOP/ERROR ICON is displayed.



---

## (DIALOG) RADIOGROUP

---

### RADIOGROUP (ControlNum, DialogIndex)

The (DIALOG) RADIOGROUP function is used with the DIALOG command. It indicates which radio button in the specified radio group is selected.

#### Arguments

##### ControlNum

The optional ControlNum argument is a value (from 1 to  $n$ ) specifying a radio group, where  $n$  is the number of radio groups in the dialog box. You do not need to include the ControlNum argument if there is only one radio group in the dialog box.

##### DialogIndex

The optional DialogIndex argument is an integer (from 0 to 15) indicating which dialog contains the radio group you want to test. If you do not include the DialogIndex argument, DCS uses the default DialogIndex value of zero.

#### Result

The result is an integer numeric value corresponding to the number of the selected radio button in the specified radio group. If the specified radio group does not exist, this function returns -1 (negative one).

#### Comments

The (DIALOG) RADIOGROUP function retrieves a result only while the specified dialog box is still active (prior to the DIALOG CANCEL command). That is, if the dialog box no longer exists, the result of this function is invalid.

 Also see: (DIALOG) RADIOGROUP command

#### Example

In this example:

```
%rad2 = RADIOGROUP (2)
```

if the third radio button in radio group two has been selected, the numeric variable %rad2 assumes the value 3.

---

# DIALOGHANDLE

---

## DIALOGHANDLE (Index)

The DIALOGHANDLE function returns the window handle of the specified dialog box.

### Arguments

#### Index

The optional `Index` argument is an integer (0 to 15) identifying a particular dialog box. Zero (0) will be used if no index is provided.

### Results

The result is an integer specifying the window handle of the window. If an error occurs, the result is a window handle of zero (0).

### Comments

If the window handle cannot be returned, the `ERROR` function is set to `TRUE`.

---

# DIRECTORY

---

## DIRECTORY (Type)

The DIRECTORY function returns the path of default directory for the specified type of file.

### Arguments

#### Type

The Type argument is one of the following keywords:

Keyword	DCS Directory	Default Path*
DNLOAD	Download directory	...DNLOAD\
DYNASYS	Executables/DLLs directory	C:\PROGRAM FILES\FUTURESOFT\DCSERIES
ICONPATH	Icon files directory	C:\PROGRAM FILES\FUTURESOFT
MEMO	Memo directory	...MEMOS\
SCRIPT	Script-text directory	...SCRIPTS\
SETTINGS	Settings directory	...SESSION\
TASK	Compiled script directory	...SCRIPTS\
TRACEPATH	Trace files directory	...TRACE\
UPLOAD	Upload directory	...UPLOAD\

\*The default path for WIN95/98 users on all directories begins with C:\PROGRAM FILES\FUTURESOFT\DCSERIES\...

\*The default path for WIN/NT users on all directories (except DYNASYSE and ICONPATH) begins with C:\WINNT\PROFILES\username\PERSONAL\...

Keyword	Windows Directory	Default Path
WINDOWS	Main directory	C:\WINDOWS\
WINSYS	System directory	C:\WINDOWS\SYSTEM\ or C:\WINNT\System32

### Result

The result is a string value containing the current default directory for the type of file as defined in the **File Locations** tab of the **Options** dialog.

### Comments

The default directory for a file type is originally specified during installation and is stored in the Windows registry. These values can be viewed and set using the **File Locations** tab of the **Options** dialog (**Tools > Options**). Also, each time DCS executes a SET DIRECTORY command, DCS changes the default directory for a file type to the specified directory.

The comment above does not apply to the directories found with the DYNASYS, WINDOWS, and WINSYS keywords. You cannot change these paths with the SET DIRECTORY command or change them in the **File Locations** tab.

---

## DIRECTORY, *continued*

---

### Example

In this example:

```
$path = DIRECTORY (MEMO)
```

if a memo was last opened in the default MEMO directory on drive C:, `$path` contains C:\PROGRAM FILES\FUTURESOFT\DCSERIES\MEMOS.

---

# DISKSPACE

---

## DISKSPACE (Drive)

The DISKSPACE function returns the number of bytes free on the specified drive.

### Arguments

#### Drive

The optional **Drive** argument is a string specifying the desired drive (the trailing colon is optional). The **Drive** argument must specify a valid drive for your system. If the drive is not specified, the default is the current working drive.

### Result

The result is an integer value indicating the number of bytes of free storage on the specified drive.

### Example

```
%free = DISKSPACE ("C:")
```

---

# ENCRYPT

---

## ENCRYPT (OrigString, Key)

The ENCRYPT function encodes a specified string using a specified encryption key.

### Arguments

#### OrigString

The OrigString argument contains the string to encode.

#### Key

The Key argument is a string which the function will use as an encryption key.

### Result

The result is a string variable containing the encoded version of the original string.

### Comments

The encryption scheme produces an encrypted string which might have significantly more characters than the original string.

To decode the result of this function, you must provide the encrypted string and the encryption key used to encrypt the string to the DECRYPT function.

 Also see: DECRYPT function

### Example

In the example below:

```
$save = ENCRYPT ("Arnold Wilson", "secret")
DISPLAY (0,0) $save
DISPLAY (1,0) DECRYPT ($save, "secret")
```

the ENCRYPT function encodes the contents of the string `Arnold Wilson`, such that the string contained in `$save` will bear no resemblance to the original string. The string `secret` is the encryption key. The first `DISPLAY` command displays the encrypted string. The second `DISPLAY` command displays the string `Arnold Wilson`, since the correct encryption key is given to the DECRYPT function.

---

## EOF

---

### EOF ( )

The EOF function is used with the RECORD READ command to indicate whether the script has attempted to read past the end of a table.

#### Arguments

The EOF function takes no arguments.

#### Result

The result is a Boolean value, where TRUE indicates the end of the table has been passed, and FALSE indicates the read was successful. If the EOF function returns TRUE, then the result of a read will not be valid.

#### Example

In this example:

```
Record Read 1 at 0
While not EOF()
Begin
Display @R1
Record Read 1
End
Display "Done"
```

DCS attempts to read the contents of table one. Each record is displayed as it is read. If the end of file is reached, EOF evaluates to TRUE. Execution then branches past the END command, and the text Done is displayed in the session window.

---

# ERROR

---

## ERROR ( )

The ERROR function indicates whether an error occurred during script execution.

### Arguments

The ERROR function takes no arguments.

### Result

The result is a Boolean value, where TRUE indicates that an error occurred during script execution, and FALSE indicates that an error did not occur during script execution.

 Also see: [RESULT](#) function

### Example

In this example:

```
File Copy "c:\config.sys" to "c:\config.bak"  
If Error()  
Begin  
Display (0,0) "Unable to copy file"  
Cancel  
End
```

DCS attempts to copy a file. If DCS is unable to copy `c:\config.sys` to `c:\config.bak`, the ERROR function returns TRUE and an error message is displayed. Otherwise, the copy operation succeeds and the script is terminated.



---

## EXFLDATTR

---

### EXFLDATTR (FieldNum, WinHandle)

The EXFLDATTR function is only valid with IBM TN3270, IBM TN5250 or Tandem 6530 emulator sessions. The EXFLDATTR function identifies the normal and extended field attributes for a field.

#### Arguments

##### FieldNum

The **FieldNum** argument is an integer and identifies a field in a session window. For information on acquiring the field number, see the FLDNUM function.

##### WinHandle

The optional **WinHandle** argument is an integer and identifies a window in DCS. This argument allows a script to check the attributes of a field in a particular session, not merely the active session window.

#### Result

For the Tandem emulation the result is an integer. However, you must resolve the result into its description with the bitwise functions: BAND, BNOT, BOR, and BXOR. The integer is composed of two bytes. Each of the bit positions have significance in determining the attributes for a field.

---

##### bits 0 - 2 - entry type

---

0 = Free entry

1 = Alpha

2 = Numeric

3 = Aphanumeric

4 = Full numeric

5 = Full numeric w/space

6 = Alpha w/space

7 = Alpha numeric w/space

---

##### bits 3 - 4 - extended data type

---

0 = normal (keyboard)

1 = upshift

2 = AID

4 = Any device (keyboard, AID)

6 = reserved

---

##### bit 6 - uses enhanced color (1 = on)

---

##### bit 7 - Protected/Unprotected (1 = protected)

---

---

## EXFLDATTR, *continued*

---

For the IBM TN3270 emulation the result is an integer. However, you must resolve the result into its description with the bitwise functions: **BAND**, **BNOT**, **BOR**, and **BXOR**. The integer is composed of two bytes. Each of the bit positions have significance in determining the attributes for a field.

High Byte		
Bit 7:	x	Not used
Bit 6:	x	Not used
Bit 5:	0	= Unprotected
	1	= Protected
Bit 4:	0	= Alphanumeric
	1	= Numeric
Bits 3, 2:	00	= Normal display, and Light pen not detected
	01	= Normal display, and Light pen detected
	10	= High intensity display, and Light pen detected
	11	= No display, and Light pen not detected
Bit 1:	0	= No extended field attributes
	1	= Extended field attributes specified
Bit 0:	0	= Not modified
	1	= Modified

Low Byte		
Bit 7:	0	= Opaque
	1	= Transparent
Bits 6,5,4:	000	= Default color
	001	= Blue
	010	= Red
	011	= Pink
	100	= Green
	101	= Turquoise
	110	= Yellow
	111	= White (neutral)
Bit 3:	0	= Normal Intensity
	1	= High Intensity
Bit 2:	0	= No Underscore
	1	= Underscore
Bit 1:	0	= No Reverse video
	1	= Reverse video
Bit 0:	0	= No Blink
	1	= Blink

---

## EXFLDATTR, *continued*

For the IBM TN5250 emulation the result is an integer. However, you must resolve the result into its description with the bitwise functions: BAND, BNOT, BOR, and BXOR. The integer is composed of two bytes. Each of the bit positions have significance in determining the attributes for a field.

High Byte			
Bits 7,6:	01		
Bit 5:	0	=	Not a Bypass field
	1	=	Bypass field
Bit 4:	0	=	Duplication not allowed in this field
	1	=	Duplication is allowed in this field
Bit 3:	0	=	Field has not been Modified
	1	=	Field has been modified
Bits 2,1,0:	000	=	Alphabetic shift
	001	=	Alphabetic only
	010	=	Numeric shift
	011	=	Numeric only
	100	=	Katakana shift
	101	=	Digits only (5294)
	110	=	I/O
	111	=	Signed numeric
Low Byte			
Bit 7:	0	=	No auto enter
	1	=	Auto enter
Bit 6:	0	=	Field exit key is not required
	1	=	Field exit key is required
Bit 5:	0	=	Accept lowercase letters
	1	=	Translate operator-entered letters to uppercase
Bit 4:		=	Reserved
Bit 3:	0	=	Not a mandatory enter field
	1	=	Is a mandatory enter field
Bits 2,1,0:	000	=	No adjust specified
	001	=	Reserved
	010	=	Reserved
	011	=	Reserved
	100	=	Reserved
	101	=	Right adjust, zero fill
	110	=	Right adjust, blank fill
	111	=	Mandatory fill

**Comments**

---

## EXFLDATTR, *continued*

---

If the **WinHandle** argument is not included in this function, the window handle for the current session will be used as the window handle argument in the function. The window handle for a particular window may be retrieved with the functions **HWNDLIST** or **WINDOWHND**, or from the **CONNECT** command.

### Example

This script:

```
%WinHandle = WindowHnd ("Scr1")
%FieldNum = FldNum (5, 20, %WinHandle)
%FieldAttrs = ExFldAttr (%FieldNum, %WinHandle)
%MaskAttrs = BNot (%FieldAttrs)

; In binary, 0x2000 is 0010.0000.0000.0000

%MaskAttrs = BOr (%MaskAttrs, 0x2000)
%MaskAttrs = BAnd (%FieldAttrs, %MaskAttrs)
If %MaskAttrs = 0
    Perform DisplayDialog ("Field " | Str (%FieldNum) |
" is unprotected.")
Else Perform DisplayDialog ("Field " | Str (%FieldNum) |
" is protected.")

Return

;*****

*DisplayDialog ($String)

Dialog "Results"
    Message $String
    Button "OK" Resume
Dialog End
Wait Resume
Return
```

looks at a field located on the fifth row and twentieth column of the session **Scr1** and determines whether the field is protected or unprotected. For purposes of illustration, the script only displays a dialog indicating its finding.

By using the bitwise functions in successive order, the script is able to determine the value of bit position thirteen. A mask is created to cover over the bits that the script is not interested in, and in this case the script is only interested in bit position thirteen. The **BNOT** function creates the initial mask, which is the negative of the integer for the field attributes. If this initial mask were used as a mask, it would cover over all the bits of the attributes number. The **BOR** function performs an OR operation on a binary number in the form of a hexadecimal number to

---

## **EXFLDATTR, *continued***

---

the mask. In this case, the **BOR** function makes sure that bit position thirteen in the mask is 1 (one). Then the **BAND** function performs an AND operation on the original attributes number with the mask. The result of this process could only be 0 (zero) or a number that is not zero. If the result is zero, the field is unprotected. If the result is not zero, the field is protected.

---

# EXISTS

---

## EXISTS (FileName)

The EXISTS function indicates whether the specified file exists.

### Arguments

#### FileName

The FileName argument is a string specifying the name of a file. The FileName argument must specify a valid file name for your system.

### Result

The result is a Boolean value, where TRUE indicates the specified file exists.

### Comments

If you do not specify a path with the file name, the default is DCS's default installation directory.

### Example

In this example:

```
IF EXISTS (DIRECTORY (MEMO) | "MEMO.TXT")
FILE RENAME DIRECTORY (MEMO) | "MEMO.TXT" \
    TO DIRECTORY (MEMO) | "MEMO.BAK"
```

If the file MEMO . TXT exists in the default MEMO directory, the EXISTS function evaluates to TRUE, and the file is renamed MEMO . BAK.

---

## FILESIZE

---

### FILESIZE (FileName)

The FILESIZE function returns the number of bytes in the specified file.

#### Arguments

##### FileName

The FileName argument is a string specifying the name of a file. The FileName argument must specify a valid file name for your system.

#### Result

The result is a numeric value indicating the size of the specified file in bytes.

#### Comments

If the file does not exist, the ERROR function returns TRUE and the result is -1.

If you do not specify a path with the file name, the default is the DCS 9 default installation directory.

#### Example

```
%size = FILESIZE ("EXCHANGE.PST")
```

---

# FILTER

---

## **FILTER (String, SearchChars, ReplaceChars)**

The FILTER function replaces characters found in the specified string.

### **Arguments**

#### **String**

The String argument specifies the string to search.

#### **SearchChars**

The SearchChars argument is a string specifying one or more characters for which to search. If more than one character is specified, DCS searches for each individual character specified, not character sequences.

#### **ReplaceChars**

The optional **ReplaceChars** argument is a string specifying one or more characters which will replace found characters. There is a one-to-one correspondence between the characters specified in the SearchChars argument and the **ReplaceChars** argument.

Characters specified in the SearchChars argument that do not have corresponding characters in the **ReplaceChars** argument are deleted. Characters specified in the **ReplaceChars** argument that do not have corresponding characters in the SearchChars argument are ignored.

### **Result**

The result is a string value containing the modified string.

### **Comments**

Both the SearchChars and the **ReplaceChars** arguments are case-sensitive.

### **Example**

In this example:

```
$file = FILTER ("\\TEST\\USER\\FILE", "\\", ":")
```

the **FILTER** function was used to convert a DOS file name to a Macintosh file name. The string variable \$file assumes the value :TEST:USER:FILE.

In this example:

```
$string = FILTER ("ABCAX0AX234", "AX0", "Zn")
```

the string variable \$string assumes the value ZBCZnZn234. All A characters are replaced with Z characters, all X characters are replaced with n characters, and all 0 (zero) characters are deleted.



---

## ***FILTER, continued***

---

In this example:

```
$string = FILTER ("ABCAX0AX234", "AX0")
```

the string variable `$string` contains the value BC234. All A, X and 0 (zero) characters are deleted.

In this example:

```
$string = FILTER ("A B C D", " ", "")
```

the string variable `$string` contains the value ABCD. All spaces are deleted.

---

# FLDATTR

---

## FLDATTR (Field, WinHandle)

The FLDATTR function is only valid with IBM TN3270, IBM TN5250, and Tandem 6530 emulation sessions. It returns the field attributes for the specified field in a session window.

### Arguments

#### Field

The **Field** argument is a numeric identifying the field.

#### WinHandle

The optional **WinHandle** argument is an integer specifying a particular session window in DCS.

### Result

For the Tandem emulation the result is an integer value specifying one of the following field attributes:

Value	Field Attribute
0	Unprotected
32	Protected

For the IBM3270 emulation The result is an integer value specifying one of the following field attributes:

Value	Field Attribute
0	Normal (alphanumeric, unprotected)
16	Numeric (unprotected)
32	Protected (alphanumeric)
48	Numeric (protected)

For the IBM TN5250 emulation the result is an integer value between hex 0x20 and 0x3F (decimal 32 – 64). This value corresponds to the IBM TN5250 screen attribute (full color).

 **Note:** If ERROR is set to TRUE, a -1 (negative one) is returned.

### Comments

If the **WinHandle** argument is not included in this function, DCS uses the window handle of the active session window.

The ERROR function returns TRUE if either **Field** or **WinHandle** is invalid.

---

## FLDATTR, *continued*

---

### Example

In this example:

```
%attrib = FLDATTR (3)
```

if field three contains numeric unprotected data, the integer variable `%attrib` assumes the value 16.

In this example:

```
%attrib = FLDATTR (2, %winhnd)
```

if the second field in the session specified by the variable `%winhnd` contains numeric protected data, the integer variable `%attrib` assumes the value 48.

---

## FLDATTREXPOS

---

### FLDATTREXPOS (Row, Column, WinHandle)

The FLDATTREXPOS function is only valid with IBM TN3270, IBM TN5250, and Tandem 6530 emulation sessions. The FLDATTREXPOS function indicates the normal and extended field attributes for a field at a specific position.

#### Arguments

##### Row

The **Row** argument is an integer specifying the row number of a field. The first row is considered row 0 (zero).

##### Column

The **Column** argument is an integer specifying the column number of a field. The first column is considered column 0 (zero).

##### WinHandle

The optional **WinHandle** argument is an integer and identifies a window in DCS. This argument allows a script to check the attributes of a field in a particular session window.

#### Result

The result is an integer. However, you must resolve the result into its description with bit-wise operators (AND, NOT, OR, and XOR). The integer is composed of two bytes. Each of the sixteen bit positions have significance in determining the attributes for a field.

 Also see: EXFLDATTR tables

#### Comments

If the **WinHandle** argument is not included in this function, the window handle for the current session will be used as the window handle argument in the function. The window handle for a particular window may be retrieved with the functions `HWNDLIST` or `WINDOWHND`, or from the `CONNECT` command.

#### Example

See the example for the `EXFLDATTR` function, substituting `FLDATTREXPOS` for `EXFLDATTR`; the major difference is the use of a row and column in the `FLDATTREXPOS` function, rather than field number used in the `EXFLDATTR` function.

---

## FLDLEN

---

### FLDLEN (Field, WinHandle)

The FLDLEN function is only valid with IBM3270, IBM5250, and Tandem 6530 emulator sessions. It returns the length of the specified field in a session window.

#### Arguments

##### Field

The **Field** argument is a numeric identifying the field.

##### WinHandle

The optional **WinHandle** argument is an integer specifying a particular session window in DCS.

#### Result

The result is a numeric value indicating the length in characters of the specified field.



**Note:** If ERROR is set TRUE a negative one (-1) is returned.

#### Comments

If the **WinHandle** argument is not included in this function, DCS uses the window handle of the current session window.

The ERROR function returns TRUE if either the **Field** argument or the **WinHandle** argument is invalid.

#### Example

In this example:

```
%length_3 = FLDLEN (3)
```

if field three begins on the first line (row zero) at the tenth column (column nine) and ends on the third line (row two) at the 19th column (column 20), and the column width is 80, the numeric variable %length\_3 assumes the value 171.

---

# FLDNUM

---

## FLDNUM (Row, Column, WinHandle)

The FLDNUM function is only valid with IBM TN3270, IBM TN5250, and Tandem 6530 emulator sessions. The FLDNUM function returns the field number, or identifier, of a field in a session window.

### Arguments:

#### Row

The **Row** argument is an integer specifying the row number of a field. The first row is considered row 0 (zero).

#### Column

The **Column** argument is an integer specifying the column number of a field. The first column is considered column 0 (zero).

#### WinHandle

The optional **WinHandle** argument is an integer and identifies a window in DCS. This argument allows a script to check the attributes of a field in a particular session, not merely the terminal window currently in focus.

The **ERROR** function returns TRUE if the **WinHandle** argument is invalid.

### Result

The result is an integer field number.

If **ERROR** is set TRUE a negative one (-1) is returned.

### Comments

If the **WinHandle** argument is not included in this function, the window handle for the current session will be used as the window handle argument in the function. The window handle for a particular window may be retrieved with the functions, **HWNDLIST** or **WINDOWHND**, or from the **CONNECT** command.

### Example

See the example for the **EXFLDATTR** function.

---

## FLDPOS

---

### FLDPOS (Field, WinHandle)

The FLDPOS function is only valid with IBM TN3270, IBM TN5250, and Tandem 6530 emulation sessions. It returns the absolute position of the beginning of the specified field in a session window.

#### Arguments

##### Field

The **Field** argument is a numeric identifying the field.

##### WinHandle

The optional **WinHandle** argument is an integer identifying a particular window containing a session in DCS.

#### Result

The result is a numeric value calculated using the following formula:

$$(\text{row} * \text{column width}) + \text{column}$$

The first line is row 0 (zero) and the first column is column 0 (zero).

If ERROR is set TRUE a negative one (-1) is returned.

#### Comments

If the **WinHandle** argument is not included in this function, DCS uses the window handle of the current session window.

The ERROR function returns TRUE if either the **Field** argument or the **WinHandle** argument is invalid.

#### Example

In this example:

```
%pos3 = FLDPOS(3)
%row = %pos3 / 80
%column = %pos3 % 80
```

if field 3 (three) starts on the second line (row one) at the tenth column (column nine), and the column width is 80, the numeric variable `%pos3` assumes value 89. The next two lines decode the `%pos` variable into the row (`%row = 1`) and column (`%column = 9`).

---

## FLDTEXT

---

### FLDTEXT (Field, WinHandle)

The FLDTEXT function is only valid with IBM TN3270 and IBM TN5250 emulation sessions. It returns the text for the specified field in a session window.

#### Arguments

##### Field

The **Field** argument is a numeric identifying the field.

##### WinHandle

The optional **WinHandle** argument is an integer identifying a particular window containing a session in DCS.

#### Result

The result is a text string.

#### Comments

If the **WinHandle** argument is not included in this function, DCS uses the window handle of the current session window.

The **ERROR** function returns TRUE if either the **Field** argument or the **WinHandle** argument is invalid.

#### Example

In this example:

```
%N = FLDNUM (2, 12)
$T = FLDTEXT (%N)
```

The variable \$T will contain all the text from the field around row 2, column 12 regardless of where the field begins and ends.



---

# GETAPPCONFIG

---

## GETAPPCONFIG (KeyString)

The GETAPPCONFIG function returns a string identifying the current state of a specified application-level setting.

### Arguments


#### KeyString

The KeyString argument must be a key string which is valid for the APPCONFIG command. The GETAPPCONFIG function will return a string identifying the current state of that setting.

The values returned by the GETAPPCONFIG function are listed in the APPCONFIG command. Also see the APPCONFIG command for a list of valid strings which may be used as KeyString arguments.

### Result

The GETAPPCONFIG function returns a string representing the current state of the setting specified by the KeyString.

 **Note:** In some instances DCS will return a 1 for TRUE and a 0 for FALSE. It is recommended that the code check for both conditions (if \$retval = TRUE or \$retval = "1 ...") to avoid possible logic flow problems.

If ERROR is set to TRUE, a null string is returned.

### Comments

The ERROR function returns TRUE if the KeyString argument is invalid.

Application configuration may be set using the **Options** dialog (select **Tools > Options**), or with the APPCONFIG command.

### Example

This script segment:

```
$bitmap = GETAPPCONFIG ("ShowBkgrdBitMap")
If $bitmap = "TRUE" APPCONFIG "ShowBkgrdBitMap=FALSE"
```

checks the current setting of the **Show Background Bitmap** option. If the setting is checked (active), it is disabled.

---

## GETCONNCONFIG

---

### GETCONNCONFIG (KeyString, WinHandle)

The GETCONNCONFIG function returns a string identifying the current state of a connector setting for the default session window.

#### Arguments

##### KeyString

The **KeyString** argument must be a key string recognized by the connector. The GETCONNCONFIG function returns a string representing the current state of that setting. For example, GETCONNCONFIG ("DataBits") would return either the string 7 or the string 8.

See the CONNCONFIG command for a list of key strings.

##### WinHandle

The optional **WinHandle** argument is an integer identifying a particular session window.

#### Result

The GETCONNCONFIG function returns a string identifying the current state of a connector setting for the session in the script's default session window.



**Note:** In some instances DCS returns a 1 for TRUE and a 0 for FALSE. It is recommended that the code check for both conditions (if \$retval = TRUE or \$retval = "1 ...") to avoid possible logic flow problems.

If ERROR is set TRUE a null string is returned.

#### Comments

The connector setting for a session window is specified on the **Connectors** tab in the **Session Properties** dialog, or with the CONNCONFIG command.

If **WinHandle** is not specified, DCS applies the connector configuration settings to the active session window.

If either the **KeyString** or **WinHandle** argument is invalid, the ERROR function returns TRUE.

#### Example

```
Set Defaultsessionhandle Active()  
$ConnDataBits = GETCONNCONFIG ("DataBits")  
%NumBits = Num ($ConnDataBits)
```

Assuming that the default session's connector is currently using a data bit setting of 8 (eight), string 8 is assigned to \$ConnDataBits. The variable %NumBits is assigned the numerical value of the string, which in this case is the numerical value 8.

If the script does not have a default session window, DCS returns a null string.

---

# GETDISPLAYCONFIG

---

## GETDISPLAYCONFIG (KeyString, WinHandle)

The GETDISPLAYCONFIG function returns a string identifying the current display settings for the default session window.

### Arguments

#### KeyString

The **KeyString** argument must be a keyword recognized by DCS for session windows. See the DISPLAYCONFIG command for a list of keywords and possible return values.

#### WinHandle

The optional **WinHandle** argument is an integer identifying a particular session window of DCS.

### Result

The GETDISPLAYCONFIG function returns a string representing the current settings of a session window.



**Note:** In some instances DCS returns a 1 for TRUE and a 0 for FALSE. It is recommended that the code check for both conditions (if \$retval = TRUE or \$retval = "1 ...") to avoid possible logic flow problems.

If ERROR is set TRUE a null string is returned.

### Comments

The display settings for a session window are set on the **Displays** tab in the **Session Properties** dialog, or with the DISPLAYCONFIG command.

If **WinHandle** is not specified, DCS retrieves the display settings for the active session window.

The ERROR function returns TRUE if either the **Field** or **WinHandle** argument is invalid.



Also see: DISPLAYCONFIG command

This script line:

```
$Cursor = GetDisplayConfig ("CursorVisible")
$DisplayInfo = $Cursor | ", " | GetDisplayConfig ("Cursor-
Type")
```

might return a string similar to the following:

1, BLOCK

This string indicates the cursor is displayed and is a block character.

---

## GETEMULCONFIG

---

### GETEMULCONFIG (KeyString, WinHandle)

The GETEMULCONFIG function returns a string identifying the current state of an emulation setting for the session in the script's default session window.

#### Arguments

##### KeyString

The **KeyString** argument must be a key string recognized by the emulation. The GETEMULCONFIG function returns a string representing the current state of that setting. For example, GETEMULCONFIG("Columns") might return either the string 80 or the string 132.

See the EMULCONFIG command for a list of key strings.

##### WinHandle

The optional **WinHandle** argument is an integer identifying a particular session window.

#### Result

The GETEMULCONFIG function returns a string identifying the current state of an emulation setting for the session in the script's default session window.



**Note:** In some instances DCS returns a 1 for TRUE and a 0 for FALSE. It is recommended that the code check for both conditions (if \$retval = TRUE or \$retval = "1 ...") to avoid possible logic flow problems.

If **ERROR** is set TRUE a null string is returned.

#### Comments

If **WinHandle** is not specified, the emulation configuration settings are applied to the active session window.

The emulation settings for a session window are specified on the **Emulations** tab in the **Session Properties** dialog, or with the EMULCONFIG command.

The **ERROR** function returns TRUE if either the **KeyString** or **WinHandle** argument is invalid.

#### Example

In this example:

```
$EmulCols = GETEMULCONFIG ("Columns")
%NumColumns = NUM ($EmulCols)
```

assuming that the default session's emulation is currently using a column width setting of 132, The string "132" is assigned to \$EmulCols. The variable %NumColumns is assigned the numerical value of the string, which in this case is the numerical value 132.

---

## GETGENERALCONFIG

---

### GETGENERALCONFIG(KeyString, WinHandle)

The GETGENERALCONFIG function returns a string identifying which options have been configured in the **General** tab of the **Session Properties** dialog for the active session.

#### Arguments

##### KeyString

The **KeyString** argument must be a key string which is valid for the GENERALCONFIG command. The GETGENERALCONFIG function returns a string representing the current state of that setting. For example, GENERALCONFIG("DisplayErrorInformation") returns a string value of either TRUE or FALSE.

See the GENERALCONFIG command for a list of key strings.

##### WinHandle

The optional **WinHandle** argument is an integer identifying a particular session window.

#### Result

The GETGENERALCONFIG function returns a string representing the current state of the setting specified by the **KeyString**.



**Note:** In some instances DCS returns a 1 for TRUE and a 0 for FALSE. It is recommended that the code check for both conditions (if \$retval = TRUE or \$retval = "1 ...") to avoid possible logic flow problems.

If ERROR is set to TRUE a null string is returned.

#### Comments

If **WinHandle** is not specified, the configuration settings are applied to the active session window.

The general settings for a session window are specified on the **General** tab in the **Session Properties** dialog, or with the GENERALCONFIG command.

The ERROR function returns TRUE if either the **KeyString** or **WinHandle** argument is invalid.

#### Example

This script segment:

```
$Autoconnect = GETGENERALCONFIG ("AutoConnect")  
If $Autconnect = "TRUE" GENERALCONFIG "AutoConnect=FALSE"
```

checks the current setting of the Auto Connect option. If the setting is checked (active), it is disabled.

---

# GETPROFILEDATA

---

## GETPROFILEDATA (Section, KeyName, INIFile)

The GETPROFILEDATA function retrieves text associated with an entry in an initialization file.

### Arguments

#### Section

The **Section** argument is a string variable that specifies the section name within an initialization file from which DCS gathers text. In an initialization file, square brackets ([ ]) surround section names.

#### KeyName

The **KeyName** argument is a string variable that specifies the entry in the section specified by the **Section** argument from which to collect text. Text is collected from an entry if the entry contains the text in the **KeyName** argument on the left side of the equal sign in the entry.

#### INIFile

The **INIFile** argument is a string variable that specifies the name of the initialization file from which DCS collects text.

### Result

If text in the initialization file is found that matches the text contained in the **KeyName** argument, the result is a string composed of the text on the right side of an entry; otherwise, the result is a null or empty string. If an error occurs, the result is a string containing the character 0 (zero).



Also see: PUTPROFILEDATA function

### Comments

An empty or null string is usually denoted by a left and right quote with nothing between the quotes. For example, after the following script line is executed the variable `$Section` contains a null string:

```
$Section = ""
```

For more information about the structure and contents of initialization files, refer to Microsoft Windows documentation.

### Example

In this example

```
$File = GETPROFILEDATA ("Child0", "IconFile", "app.ini")
```

the function searches the `app.ini` initialization file. All entries in the section `Child0` are searched for the text `"IconFile"`. If an entry contains the search text to the left of the equal sign, the variable `$File` contains the text to the right of the equal sign. If not found, `$File` contains a null string.

---

# GETXFERCONFIG

---

## GETXFERCONFIG (KeyString, WinHandle)

The GETXFERCONFIG function returns a string identifying the current state of a file transfer protocol setting for the default session window.

### Arguments

#### KeyString

The **KeyString** argument must be a key string recognized by the file transfer protocol. The GETXFERCONFIG function returns a string representing the current state of that setting. For example, GETXFERCONFIG("BlockSize") might return either the string "256" or the string "Auto".

See the XFERCONFIG command for a list of key strings.

#### WinHandle

The optional **WinHandle** argument is an integer identifying a particular child window of DCS.

### Result

The GETXFERCONFIG function returns a string identifying the current state of a file transfer protocol setting for the session in the script's default session window.



**Note:** In some instances DCS will return a 1 for TRUE and a 0 for FALSE. It is recommended that the code check for both conditions (if \$retval = TRUE or \$retval = "1 ...") to avoid possible logic flow problems.

If ERROR is set to TRUE a null string is returned.

### Comments

If **WinHandle** is not specified, DCS applies the file transfer protocol configuration settings to the active session window.

The file transfer protocol settings for a session are specified on the **File Transfers** tab in the **Session Properties** dialog, or with the XFERCONFIG command.

The ERROR function returns TRUE if either the **KeyString** or **WinHandle** argument is invalid.

---

## GETXFERCONFIG, *continued*

---

### Example

In this script:

```
$XFerBlock = GETXFERCONFIG ("BlockSize")
If $XFerBlock <> "Auto"
%Block = Num ($XFerBlock)
Else Display (0,0) $XFerBlock
```

assuming the default session's file transfer protocol is currently using a blocksize setting of 256, the string "256" is assigned to \$XFerBlock. The variable %Block is assigned the numerical value of the string, which in this case is the numerical value 256. If the string is "Auto", the string is shown in the active session window.

If the script does not have a default session window, a null string is returned.



---

# HWNDLIST

---

## HWNDLIST (NumWin)

The HWNDLIST function returns a string containing the window handles of all open child windows.

### Arguments

#### NumWin

The optional **NumWin** argument is an integer expression specifying the number of window handles to return, where the most recently activated window is specified by the integer one, the two most recently activated windows are specified by the integer two, and so on.

### Result

The result is a string composed of the handles of DCS child windows. The first handle is the most recently activated window, followed by the next most recently activated, and so on. The string is a hexadecimal representation of a Window handle and has the generalized form of 0x\*\*\*\*, where an asterisk is a character from the set (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F). Each window's handle, except for the last handle in the resulting string, is followed by a comma.

A value of -1 (negative one) in the **NumWin** argument returns a string of all child window handles. This is the default value.

A value of 0 (zero) in the **NumWin** argument returns a string of the application window handle if a child window is open in DCS. If no child window is open, a value of zero returns a null string.

Because the value returned is a string, you must use the NUM function to change the string to a numeric representation if you wish to use the result as a numeric value with other functions or commands.

### Comments

If a child window is hidden by the WINDOW HIDE command, its handle is not available via this function.

### Example

In this example

```
$wins = HWNDLIST (-1)
DISPLAY $wins
```

the HWNDLIST function returns the window handles for all of the child windows of the DCS application window.

When the display command is executed, DCS might display the string 0x3084,0x8D88,0x30D8 in the session window.

---

## HWNDLIST, *continued*

---

### Example

In this example:

```
$AllWindows = HWNDLIST (-1)
%Comma = POS ($AllWindows, ",", 1)
%WinHnd = NUM (SUBSTR ($AllWindows, 1, (%Comma -1)))
Window Activate %WinHnd
```

the hexadecimal values of all windows are assigned to `$AllWindows`. The integer variable `%WinHnd` is assigned a numeric value by changing a substring of `$AllWindows` into a numeric value with the `NUM` function. This allows DCS to use `%WinHnd` as the window handle for the `WINDOW ACTIVATE` command. Without the use of the `NUM` function, the string retrieved by the `HWNDLIST` function cannot be used as a numeric value in another function or command.

---

# ICONIC

---

## ICONIC (WinHandle)

The **ICONIC** function indicates whether a child window or the DCS application window is minimized.

### Arguments

#### WinHandle

The optional **WinHandle** argument is a numeric expression specifying a particular window. Use a **WinHandle** argument of zero (0) to specify the DCS application window.

### Result

The result is a Boolean value, where **TRUE** indicates the window is minimized, and **FALSE** indicates that the window is not minimized.

### Comments

If the **WinHandle** argument is not included in this function, DCS uses the window handle of the current session window.

The **ERROR** function returns **TRUE** if the **WinHandle** argument is not valid.

### Example

In this example:

```
%h = NUM (hwndlist (0))
IF ICONIC (%h)
WINDOW MAXIMIZE (%h)
```

**%h** contains the handle for the DCS application window. If DCS is running in a minimized window, the **ICONIC** function evaluates to **TRUE** and the DCS application window is maximized.

---

# INT

---

## INT (RealNum)

The INT function converts a real numeric value or expression to an integer value.

### Arguments

#### RealNum

The RealNum argument contains the real numeric value or expression to be converted.

### Result

The result is an integer value corresponding to the whole number portion of the RealNum argument.

### Comments

This function is most useful for converting a real value to an integer, although either an integer or a real number may be specified. All decimal places in a real number are truncated, not rounded.

### Example

In this example:

```
%num = INT (23.6)
```

the numeric variable %num assumes the value 23.

In this example:

```
DISPLAY (0,0) STR (INT (10 / 3))
```

This command directs DCS to display the string representation of the integer returned by the INT function. Since the result of the integer division is a real number (3.33333), the INT function is used to convert it to an integer. The string 3 will be displayed in the active session window.

---

# LENGTH

---

## LENGTH (String)

The LENGTH function returns the number of characters in the specified string.

### Arguments

#### String

The String argument specifies the string whose length is to be determined.

### Result

The result is an integer value indicating the number of ASCII characters, including spaces, in the specified string.

### Comments

Since a string may have a maximum length of 254 characters, this function returns a maximum value of 254.

### Example

In this example:

```
%len = LENGTH ($name)
```

if the string variable \$name contains the string "TEXAS", the numeric variable %len assumes the value "5".

If the string variable \$name contains the string "HOUSTON, TEXAS", the numeric variable %len assumes the value "14".

---

## (MENU) CHECKED

---

### CHECKED (Popup, Item)

The (MENU) CHECKED function indicates whether the specified menu item is checked or unchecked.

#### Arguments

##### Popup

The **Popup** argument is an integer specifying a popup menu, where the application control menu is specified by the integer 0 (zero) and the first popup menu is specified by the integer 1 (one).

##### Item

The **Item** argument is an integer specifying an item in the menu specified by the **Popup** argument. The first item is specified by the integer 1 (one) and both items and separators are counted. For example, the last item in a menu containing ten items and two separators is specified by the integer “12”.

#### Result

The result is a Boolean value, where TRUE indicates that the item is checked and FALSE indicates that the item is unchecked.

If either the **Popup** argument or **Item** argument is invalid, the ERROR function returns TRUE.

#### Example

In this example:

```
IF CHECKED (2, 1)
MENU UPDATE 2 1 UNCHECKED
ELSE
MENU UPDATE 2 1 CHECKED
```

the (MENU) CHECKED function returns the state of the first item in the second menu. The MENU UPDATE command toggles the state between checked and unchecked.

---

## (MENU) ENABLED

---

### ENABLED (Popup, Item)

The (MENU) ENABLED function indicates whether the specified menu item is enabled or disabled.

#### Arguments

##### Popup

The **Popup** argument is an integer specifying a popup menu, where the application control menu is specified by the integer 0 (zero) and the first menu is specified by the integer 1 (one).

##### Item

The **Item** argument is an integer specifying an item in the menu specified by the **Popup** argument. The first item is specified by the integer one and both items and separators are counted. For example, the last item in a menu containing ten items and two separators is specified by the integer “12”.

#### Result

The result is a Boolean value, where TRUE indicates that the menu item is enabled and FALSE indicates that the menu item is disabled.

If either the **Popup** argument or **Item** argument is invalid, the **ERROR** function returns TRUE.

#### Example

In this example:

```
IF ENABLED (2, 2)
MENU UPDATE 2 2 GRAYED
```

the (MENU) ENABLED function returns the state of the second item in the second menu. If the menu item is enabled (the (MENU) ENABLED function returns TRUE), the **MENU UPDATE** command disables the menu item.

In this example:

```
IF ENABLED (2, 2)
MENU UPDATE 2 2 GRAYED
ELSE
MENU UPDATE 2 2 ENABLED
```

the **ENABLED** function returns the state of the second item in the second menu. The **MENU UPDATE** command toggles the menu item between an enabled and disabled state.

---

## NETID

---

### NETID ( )

The NETID function returns the current value of the NetworkID system variable.

#### Result

The result is the current value of the NetworkID system variable. The string can contain alphanumeric characters.

#### Comments

Compare with the SET NETID command.

#### Example

In this example:

```
WHEN STRING "@"  
SEND NETID ( )
```

the result string of the NETID function is sent to the remote system when the string "@" is received through the COM port of the computer.

In this example:

```
WAIT STRING "Network?"  
BEGIN  
SEND NETID()  
END
```

the result string of the NETID function is sent to the remote system when the string "Network?" is received through the COM port of the computer.



---

## NEXT

---

### NEXT ( )

The NEXT function retrieves the next file name from the most recent list of files established using the ROUTE function.

#### Arguments

The NEXT function takes no arguments.

#### Result

The result is a string value containing the next file name in the current route.

#### Comments

If the route is empty or has not been defined, the ERROR function returns TRUE and the result is a null string.

#### Example

See the ROUTE function.

---

# NUM

---

## NUM (String)

The NUM function returns the real numeric value of the specified string.

### Arguments

#### String

The String argument contains the string to be converted to a numeric.

### Result

The result is a real numeric value corresponding to the numeric value of the String argument.

### Example

In this example:

```
!real = NUM ("111.89")
```

the real numeric variable `!real` assumes the value "111.89".

In this example:

```
%int = NUM ("123.67")
```

the integer numeric variable `%int` assumes the numeric value "123". Although the NUM function returns a real numeric, in this example it is stored in an integer numeric and, therefore, the fractional portion of the value is truncated.

---

# ORD

---

## ORD (String)

The ORD function converts the characters specified in the string to an integer numeric value.

### Arguments

#### String

The String argument specifies the string to be converted. If multiple characters are specified, the ORD function returns the sum of the ordinal values of the characters.

### Result

The result is an integer numeric value corresponding to the sum of the ASCII numeric equivalents of the characters in the String argument.

### Comments

For more information on character representations, see the appendix titled “Character Sets” in the DCS 9 Online Reference.

### Example

In this example:

```
%num = ORD ("A")
```

the numeric variable %num assumes the value “65”.

In this example:

```
%num_sum = ORD ("ABC")
```

the numeric variable %num\_sum assumes the value “198”. It returns the same value as the expression `ORD ("A") + ORD ("B") + ORD ("C")`; the sum of this expression is “65+66+67”, or “198”.

---

## PASSWORD

---

### PASSWORD ( )

The PASSWORD function returns the current value of the Password system variable.

 Note: This function does not apply to the IBM TN3270 emulation.

### Result

The result is the current value of the Password system variable. The string may contain alphanumeric characters.

### Comments

 Also see: SET PASSWORD command

### Example

In this example:

```
IF PASSWORD ( ) = "secret"  
  DISPLAY "shhhh"
```

the string "shhhh" is displayed if the Password system variable is set to "secret".

---

# PHONENUMBER

---

## PHONENUMBER ( )

The PHONENUMBER function returns the current value of the PhoneNumber system variable.

### Result

The result is the current value of the PhoneNumber system variable. The string may contain alphanumeric characters.

### Comments



Also see: SET PHONENUMBER command

### Example

In this example:

```
$oldNum = PHONENUMBER ( )
```

the string variable \$oldNum assumes the value of the PhoneNumber system variable.

---

# POS

---

## POS (String, Keyword, Start)

The POS function returns the position of the first character of a keyword within a specified string.

### Arguments

#### String

The **String** argument specifies the string in which to search for the keyword.

#### Keyword

The **Keyword** argument is a string containing the characters for which to search.

#### Start

The optional **Start** argument is a numeric specifying the character position within the specified string (from 1 to *n*) at which to begin searching. If the **Start** argument is not included, the search defaults to the first character in the string.

### Result

The result is a numeric value indicating the position of the first character of the keyword within the string. If the string does not contain the keyword, the result is the value zero.

### Example

In this example:

```
%position = POS ("DYNACOMM", "A")
```

The numeric variable `%position` assumes the numeric value "4".

---

## POSITION

---

### POSITION (WinHandle, Boolean)

The POSITION function returns a string containing the position and dimensions of the window specified by WinHandle.

#### Arguments

##### WinHandle

The WinHandle argument is a numeric expression specifying a particular window. The handle of the DCS application window is specified by the handle zero.

##### Boolean

The optional Boolean argument causes the function to return the position of the window frame when it evaluates to FALSE. When it evaluates to TRUE, the position of the window's client area is returned. The default state is FALSE.

#### Result

The result is a string value listing the coordinate position and dimensions of the specified window, in the form:

`x,y,w,h`

where (x,y) are the coordinates of the top left corner of the specified window, (w) is the window's width, and (h) is the window's height.

#### Comments

If the WinHandle argument is not valid, the ERROR function returns TRUE and the result is a null string.

Many of the WINDOW commands make use of window position coordinates.

#### Example

In this example

```
%winnum = NUM ($winstring)
$place = POSITION (%winnum)
DISPLAY $place | ``^m``
```

\$winstring is a window handle which has been parsed from a string returned by the HWNDLIST function. The string, "\$winstring", is converted to an integer and the integer, %winnum, is used as the window handle argument in the POSITION function. The resulting string is displayed with a carriage return. The string "\$place" represents the position of the window's frame.

---

# POWER

---

## POWER (Base, Exponent)

The POWER function raises the base to the specified power.

### Arguments

#### Base

The Base argument is a numeric specifying the mathematical base.

#### Exponent

The Exponent argument is a numeric specifying the power to which the base will be raised.

### Result

The result is a real numeric value corresponding to the Base raised to the Exponent.

### Example

In this example:

```
%num = POWER (3, 2)
```

the numeric variable %num assumes the value 9.

In this example:

```
!sqrt = POWER (%num, .5)
```

the real numeric variable !sqrt assumes the value equal to the square root of %num.



---

## PRTMETRICS

---

### PRTMETRICS ()

The PRTMETRICS function returns the current print parameters.

#### Arguments

The PRTMETRICS function takes no arguments.

#### Result

The result is a string value in the following format:

Columns,Lines;Font,PointSize

#### Comments

If a print channel is open, PRTMETRICS returns the current print parameters (see the PRINT OPEN command).

If no print channel is open, PRTMETRICS returns a null string.

#### Example

In this example:

```
$prt = PRTMETRICS ()
```

if the print parameters include a page size of 80 columns and 66 lines, with a 12 point Courier printer font, \$prt assumes the string value "80,66,Courier,12".

---

# PUTPROFILEDATA

---

## PUTPROFILEDATA (Section, KeyName, Data, File)

The PUTPROFILEDATA function creates a new entry or changes an existing entry within an initialization file and returns a string value to indicate whether this succeeded.

### Arguments

#### Section

The **Section** argument is a string variable containing the section name within an initialization file to which DCS writes text. In an initialization file, square brackets ( [ ] ) surround a section name.

#### KeyName

The **KeyName** argument is a string variable specifying which entry in the section indicated by the **Section** argument to change or create. DCS changes an entry if an entry contains the text in the **KeyName** argument on the left side of the equal sign in the entry. If DCS cannot find the text from the **KeyName** argument in an entry, DCS will create a new entry.

#### Data

The **Data** argument is a string variable specifying the text for an entry. The contents of the **Data** argument appears to the right of the equal sign in an entry. When DCS creates an entry, the entry will have the following format:

KeyName = Data

#### File

The **File** argument is a string variable specifying the name of the initialization file in which DCS will create or change text.

### Result

If DCS was able to change or create text in the initialization file, the result is a string containing the character 1. If an error occurs, the result is a string containing the character 0 (zero). One possible error might be a file error (such as Disk Full), which could occur when DCS tries to close the initialization file.



Also see: GETPROFILEDATA function

---

## PUTPROFILEDATA, *continued*

---

### Comments

An empty or null string is usually denoted by left and right quotation marks with nothing between them. For example, after the following script line is executed, the variable `$Section` will contain a null string:

```
$Section = ""
```

### Example

This script example:

```
$Status = PUTPROFILEDATA ("Child0", "Magic_Number", "42",  
    "")
```

searches the default application initialization file for the text "Magic\_Number" in the section Child0. If the text is found, the characters "42" are written on the right side of the equal sign of the entry. If the text is not found, an entry similar to the following is created:

```
Magic_Number=42
```

The function returns a string which is placed in `$Status`. If the operation was successful, the variable `$Status` contains the character "1" (one). If the initialization file is not changed, `$Status` contains the character "0" (zero).

---

# RANDOM

---

## RANDOM (Range)

The RANDOM function returns a random number from 0 (zero) to (Range -1), inclusive.

### Arguments

#### Range

The Range argument is an integer specifying the upper limit of the range from which the random number is generated.

### Result

The result is an integer value.

### Example

In this example:

```
%number = RANDOM (50) + 1
```

the numeric variable `%number` assumes a random value from 1 - 50, inclusive.

---

# REAL

---

## REAL (Numeric)

The REAL function returns a real value corresponding to the specified numeric.

### Arguments

#### Numeric

The Numeric argument specifies the numeric value to be converted.

### Result

The result is a real numeric value corresponding to the specified numeric value.

### Comments

This function is most useful for converting an integer value to a real value, although either an integer or a real value may be specified.

### Example

In this example:

```
!num = REAL (10)
DISPLAY (0,0) STR (!num, 4)
```

These commands direct DCS to display the string “10.0000” in the session window.

---

# RESULT

---

## RESULT ( )

The RESULT function returns the contents of the Result system variable.

### Arguments:

The RESULT function takes no arguments.

### Result

The result is a string value corresponding to the contents of the Result system variable.

### Comments

The contents of the Result system variable may be modified by several methods. The SET RESULT command may be used to assign a value to the Result system variable. Throughout the command reference, you will also see notations of commands, such as TABLE COPY, that assign a value to the Result system variable. If a task error occurs, the Result system variable is assigned a value equivalent to the error message.



Also see: SET RESULT command  
ERROR function

### Example

This set of commands:

```
TABLE COPY 0 TO 1 INCLUDE "a"  
IF NUM (RESULT ()) > 0  
DISPLAY "More records"
```

instructs DCS to copy to table one the records from table zero that begin with the character a. The TABLE COPY command assigns the number of copied records to the Result system variable.

---

# ROUND

---

## ROUND (Real, Places, Boolean)

The ROUND function rounds the decimal places in the specified real numeric.

### Arguments:

#### Real

The **Real** argument is a real numeric specifying the number to be rounded.

#### Places

The optional **Places** argument is a numeric specifying the number of decimal places to which to round the real numeric. If the **Places** argument is not included, DCS rounds the real numeric to the number of decimal places specified in the most recently executed SET DECIMAL command. If no SET DECIMAL command has been executed, the default value of the **Places** argument is zero.

#### Boolean

The optional **Boolean** argument specifies whether to round or truncate the specified real numeric. If **Boolean** evaluates to TRUE, the numeric is rounded. If **Boolean** evaluates to FALSE, the numeric is truncated. If not included, the default value of the **Boolean** argument is TRUE.

### Result

The result is a real numeric value rounded to the specified decimal place.

### Comments

For decimal values of 0, 1, 2, 3, and 4, the numeric is rounded down. For decimal values of 5, 6, 7, 8, and 9, the numeric is rounded up.

### Example

In this example:

```
!pi = ROUND (3.141592654, 2, TRUE)
```

a copy of the **REAL** argument in the **ROUND** function is rounded down to the second decimal place, and the variable `!pi` assumes the value “3.14”.

---

# ROUTE

---

## ROUTE (FileSpec, ATTRIBUTES Type)

The ROUTE function creates a list of files which match the file specification.

### Arguments

#### FileSpec

The FileSpec argument is a string specifying the type of files to include in the route.

FileSpec argument syntax:

FileSpec = drive:\path...\FileName.extension

A FileSpec argument must contain at least one of the parts shown above, and can contain the wildcard characters (?) and (\*). The (?) wildcard indicates that any single character can occupy that string position. The (\*) wildcard indicates that zero or more characters can occupy that string position.

#### ATTRIBUTES Type

The optional **ATTRIBUTES** clause indicates which files to include in the route. The **Type** argument is a numeric specifying the file attributes (see also the ATTRIBUTES function).

Value	File Attribute	Standard File
1	Read-Only File	yes
2	Hidden File	no
4	System File	no
16	Subdirectory	no
32	Archived File	yes
64	Compressed File	yes

To specify multiple attributes, use the sum of all of the attribute values as the **Type** argument.

If the **ATTRIBUTES** clause is not included, or is zero, only standard files will be included in the resulting route. Otherwise, only those files whose attributes have been specified will be included in the route.



Note: This differs from previous versions of DCS, in which standard files were also included whether or not their attributes were specified. If you are using a script written under DCS Asynchronous, DCS/Elite or a DCS OpenConnect product, you will need to add the values of the standard files to any **Type** arguments in the script.

### Result

The result is a string value specifying the first file name in the argument FileSpec with the attributes described in the **ATTRIBUTES** clause. Succeeding files may be found with the **NEXT** function.



---

## ROUTE, *continued*

---

### Comments

If no files match, the `ERROR` function returns `TRUE` and the result is a null string.



Also see: `NEXT` function

### Example

This set of commands:

```
$pathname = "C:\"
$nextfile = ROUTE($pathname | '*.*', 127)
$nextfile = NEXT()
While ($nextfile != "")
BEGIN
    display "^M" | " =>"
    display $nextfile
    $nextfile = NEXT()
END
```

establishes a route through all the directories and files in the root directory (the sum 127 is the **Type** argument and includes the values of all standard files), and displays their names. The `WHILE` loop is designed to continue displaying the next file until the route is empty.

---

# SCREEN

---

## SCREEN (Row, Column, Length, WinHandle)

The SCREEN function retrieves a line or portion of a line of data at the specified position in a session window.

### Arguments

#### Row

The **Row** argument is an integer specifying the row number of the desired data. The first row is considered row zero.

For those emulations which display data in a status line, the data can be captured from the top and bottom status lines by using the following integers in the **Row** argument:

Status Line	Integer
Top	-1
Bottom	-2

#### Column

The optional **Column** argument is an integer specifying the column number of the desired data. The first column is considered column zero. If the **Column** argument is not included, a default column offset of zero is assumed.

#### Length

The optional **Length** argument is a numeric specifying the length (in characters) of the desired data. If the **Length** argument is not included, or if **Length** exceeds the length of the line, the remainder of the line is collected.

If you use the **Length** argument, you must also include the **Column** argument. If you do not include the **Column** argument, the script will compile, but the SCREEN function may not perform as expected.

```
SCREEN (1, 3, 5, 0x4BD3) (Row, Column, Length, & WinHandle arguments)
```

*or*

```
SCREEN (1, 3) (Row & Column arguments only)
```

*but not*

```
SCREEN (1, , 5) (incorrect usage)
```

#### WinHandle

The optional **WinHandle** argument is an integer specifying a particular window in DCS. The inclusion of this argument allows a script in DCS to retrieve a string from a particular session window of DCS.

---

## SCREEN, *continued*

---

### Result

The result is a string value containing the window data within the specified area.

### Comments

If the **WinHandle** argument is not included in this function, DCS uses the window handle of the active session window.

The **ERROR** function returns **TRUE** if the **WinHandle** argument is not valid.

### Examples

In this example:

```
$line = SCREEN (0)
```

the string variable `$line` assumes the string value of the first line of data on the screen.

In this example:

```
$chars = SCREEN (1, 3, 7)
```

the string variable `$chars` assumes the string value of seven characters of screen data, starting at the second row and fourth column.

---

# SEARCH

---

## SEARCH (Row, Column, Length, String, WinHandle)

The SEARCH function returns the absolute position of the first character of the specified string in the specified session window.

 Caution! This function is case sensitive.

### Arguments

#### Row

The optional **Row** argument is an integer specifying the row offset for the starting position from which the search is performed. The first row is considered row 0 (zero). If the **Row** argument is not included, the first row will be the starting row of the search.

#### Column

The optional **Column** argument is an integer specifying the column offset for the starting position. The first column is column 0 (zero). If the **Column** argument is not included, the first column will be the starting column of the search.

If you use the **Column** argument, you must also include the **Row** argument.

#### Length

The optional **Length** argument is an integer specifying the number of characters in the window to be included in the search. If the **Length** argument is not included, all remaining characters in the window are examined.

If you use the **Length** argument, you must also include the **Row** and **Column** arguments. If they are not included, the script compiles, but the SEARCH function may not perform as expected.

```
SEARCH ("Texas")  
(String argument only)
```

*or*

```
SEARCH (1, 3, 5, "Texas", 0x3AFF)  
(Row, Column, Length, String, & WinHandle arguments)
```

*or*

```
SEARCH (1, 3, "Texas")  
(Row, Column, & String arguments only)
```

*but not*

```
SEARCH (1, , 5, "Texas")  
(incorrect usage)
```

---

## SEARCH, *continued*

---

### String

The **String** argument specifies the string value for a case-sensitive search.

### WinHandle

The optional **WinHandle** argument is an integer identifying a particular window in DCS. The inclusion of this argument allows a script in DCS to search a particular window.

### Result

The result is an integer value that is calculated using the following formula:

$$(\text{row} * \text{column width}) + \text{column}$$

The first row is row zero and the first column is column zero. If the string is not found, the result is -1.

### Comments

If the **WinHandle** argument is not included in this function, DCS uses the window handle for the active session window.

The **SEARCH** function does *not* search the status line.

The **ERROR** function returns TRUE if the **WinHandle** argument is *not* valid.

### Example

In this example:

```
%position = SEARCH (0, 0, "USER ID:")
```

if the string **USER ID:** begins on the second line (row one) at the fifth column (column four), and the column width is 80, **%Position** assumes the value "84". This value is derived using the formula above as follows:  $(1 * 80) + 4 = 84$ .

---

# SEARCHINRECT

---

## SEARCHINRECT (TopRow, BottomRow, LeftCol, RightCol, String, WinHandle)

The SEARCHINRECT function searches for a specified string within a specified rectangular screen area. It then returns the position of the first character of the string.

### Arguments

#### TopRow, BottomRow, LeftCol, RightCol

The TopRow, BottomRow, LeftCol, and RightCol arguments are integers that define a region of the session screen, where 0,0 is the upper left character position of the screen, and where the lower right character position of the screen is specified by the maximum number of rows and columns for the emulation (MaxRowsInEmulation, MaxColsInEmulation). If you supply a TopRow argument or a LeftCol argument that is less than zero, the function will use zero as the value of the argument. Similarly, if a BottomRow argument is greater than MaxRowsInEmulation, the function will use MaxRowsInEmulation as the value of the argument, and if the RightCol argument is greater than MaxColsInEmulation, the function will use MaxColsInEmulation as the value of the argument.

#### String

The String argument is the specified string for which DCS is to search.

#### WinHandle

The optional WinHandle argument is an integer identifying a particular window in DCS. The inclusion of this argument allows a script in DCS to search a particular window.

### Result

When the function finds the whole string in the defined area, the function returns the position of the first character. The position is calculated using the following formula:

$$(\text{row} * \text{column width}) + \text{column}$$

The first row is row zero and the first column is column zero. If the string is not in the area defined by the function (or if the string is only partially in the area), the function returns -1 (negative one).

### Example

In this example:

```
SEARCHINRECT (10,15,40,80,"User ID:")
```

a rectangular screen area is searched beginning at row 10, column 40, and ending at row 15, column 80. If the first character of the string `User ID:` is in the twelfth line (row 11) at the sixtieth column (column 59) of the session screen, and if the terminal emulation has a column width of 132, the function returns 1511, or mathematically,  $(11 * 132) + 59$ .

---

# SECONDS

---

## SECONDS (Time, Date)

The SECONDS function returns the number of seconds between January 1, 1904 and the time and date indicated by the **Time** and **Date** arguments.

### Arguments

#### Time

The optional **Time** argument is a string specifying the time to which the function should calculate the number of seconds. The **Time** argument must be in the time format indicated by your Windows initialization file or registry. The TIME function also provides the time in the proper format.

If you do not include the **Time** argument, DCS will use 00:00:00 AM (midnight).

#### Date

The optional **Date** argument is a string specifying the date to which the function should calculate the number of seconds. The **Date** argument must be in the date format specified in your Windows initialization file or registry. The DATE function also provides the date in the proper format.

If you do not include the **Date** argument, DCS uses January 1, 1904. If you include the **Date** argument, you must include the **Time** argument.

### Result

The result is an integer representing number of seconds elapsed from 12:00:00 AM, January 1, 1904 to the specified time and date. The result may be a large integer, and due to the sign extension, it may display as a negative number. The result of the function is best used in comparisons or as an argument for the DATE and TIME functions.

### Example

In this example:

```
$tomorrow = DATE (SECONDS (TIME (), DATE ()) + 3600 * 24)
```

The string variable \$tomorrow assumes the string value of the date, exactly 24 hours from when the script line executes. The innermost DATE function returns the current date. The SECONDS function then converts this date to seconds. The number of seconds from the SECONDS function is added to the number of seconds in one day. Finally, the DATE function transforms the summation of seconds to the date of the next and assigns the result to \$tomorrow.

This command:

```
IF SECONDS (DATE ()) >= SECONDS ($target_date)
PERFORM update
```

compares the current date with the target date specified by \$target\_date and then branches execution accordingly.

---

# SETTINGS

---

## SETTINGS (Keyword, WinHandle)

The SETTINGS function returns values of the currently selected session properties.

### Arguments

#### Keyword

The Keyword argument must be one of the keywords below.

Keyword	Description
BACKSPACEKEY	For use only with the VT Series emulation.  Returns either TRUE or FALSE, indicating whether the [BACKSPACE] key is configured to operate as Delete (FALSE) or Backspace (TRUE).  Default setting: Delete (FALSE)
BAUDRATE	Returns the current setting for the baud rate as a numeric string.
BINARYTRANSFERPARAMS	Returns a string. The syntax of the string depends on which binary transfer protocol you have selected.  KERMIT  If the current protocol is Kermit, the ERROR string is returned. Parameters for this protocol cannot be obtained by this function.

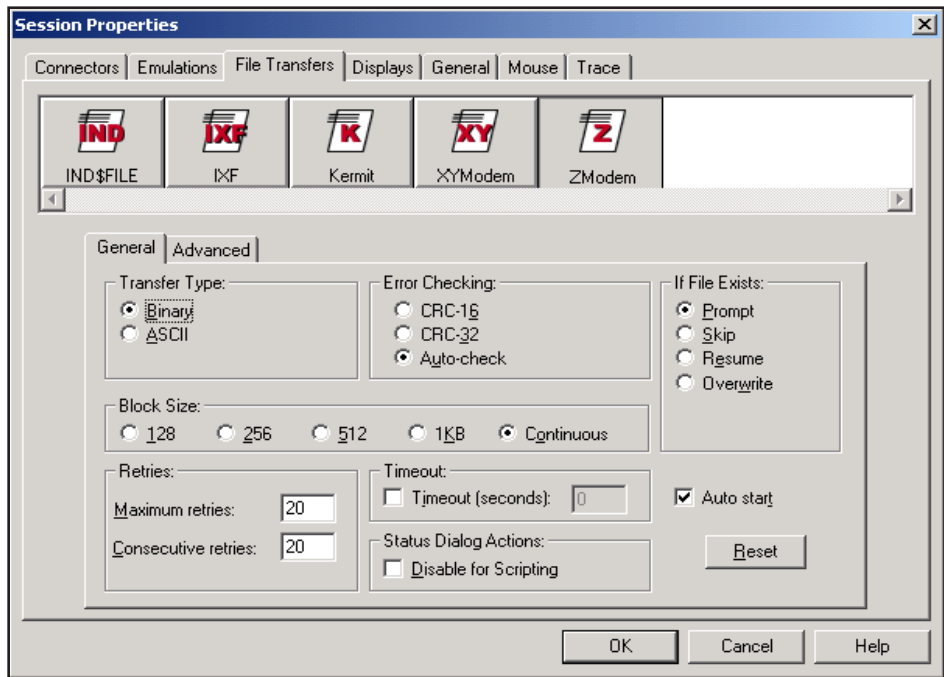
---



## SETTINGS, *continued*

Keyword	Description
BINARYTRANSFERPARAMS, <i>continued</i>	
	ZModem
	If the current protocol is ZModem, a string is returned where each string element refers to a <b>ZModem Settings</b> dialog control. To display this dialog, open DCS and select <b>File &gt; Session &gt; Properties &gt; File Transfers tab &gt; ZModem</b> . Figure 2.1 shows an example of this dialog.

**Figure 2.1**  
General ZModem settings



## SETTINGS, *continued*

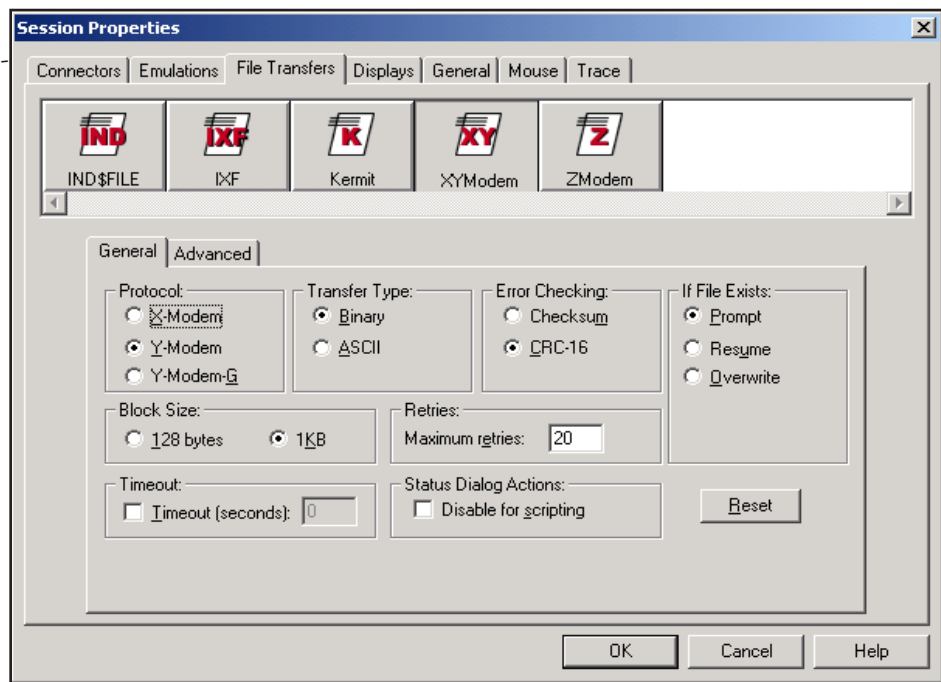
Keyword	Description
BINARYTRANSFERPARAMS, <i>continued</i>	
ZModem, <i>continued</i>	
Returned String:	
ZMODEM; PACKETSIZE Packet; TIMING Timing; ERRORCHECK Error; RETRIES Max MAX Cons CONS; YIELD Yield; Exist EXISTING; AUTO- START Enabled	
Where:	

String Word	Description
Packet	Represents size of transfer packet. Valid values: 1K, 512, 256, 128, AUTO
Timing	Represents time-out option. Valid values: STANDARD, TIGHT, LOOSE
Error	Represents the ERROR checking method. Valid vaues: AUTO, CRC32, CRC16
Max	Represents the maximum number of retries to allow. Valid values: Numeric character string
Cons	Indicates the maximum number of consecutive retries to allow. Valid values: Numeric character string
Yield	Represents the number of packet to send before yielding to other Windows processes. Valid values: Numeric character string The higher the numeric value, the longer DCS controls Windows' resources.
Exist	Represents the action to take when the file name to transfer matches a file name on the receiving system. Valid values: RESUME, PROMPT, OVERWRITE, SKIP, RENAME
Enabled	Indicates to/not to automatically transfer files when a sending system initiates a transfer. Valid values: ENABLED, DISABLED

## SETTINGS, *continued*

Keyword	Description
BINARYTRANSFERPARAMS, <i>continued</i>	
	XModem or YModem
	If the current protocol is XModem or YModem, a string is returned where each string element refers to a <b>XYModem Settings</b> dialog control. To see this dialog, select <b>File &gt; Session &gt; Properties &gt; File Transfers tab &gt; XYModem</b> .

**Figure 2.2**  
General XYModem settings



## SETTINGS, *continued*

Keyword	Description
BINARYTRANSFERPARAMS, <i>continued</i>	
XMODEM or YMODEM, <i>continued</i>	
Returned string:	
Option; PACKETSIZE Packet; TIMING Timing; ERRORCHECK Error; RETRIES Max MAX Cons CONS; YIELD Yield; Size	
Where:	
String Word	Description
Option	Specifies the XModem or YModem protocol option. Valid values: XMODEM, YMODEM, YMODEM251, YMODEM-G
Packet	Specifies the size of the transfer packet. Valid values: 1K, 128, AUTO
Timing	Specifies the time-out option. Valid values: STANDARD, TIGHT, LOOSE
Error	Specifies the ERROR checking method. Valid values: CRC or CHECKSUM
Max	Specifies the maximum number of retries to allow. Valid values: Numeric character string
Cons	Specifies the maximum number of consecutive retries to allow. Valid values: Numeric character string
Yield	Specifies the number of packets to send before it yielding to other Windows' processes. Valid values: Numeric character string The higher the numeric value, the longer DCS controls Windows' resources.
Size	Specifies to/not to pad the last file packet. Valid values: KEEPSIZE (maintain the size of the original file) NOKEEPSIZE (pad the last packet of the file with null characters)

---

## SETTINGS, *continued*

---

Keyword	Description
BINARYTRANSFERS	Depending on the current file transfer protocol selected, returns: IND\$file*, IXF*, Kermit, XModem, YModem Batch, or ZModem. (*only available with a client option)
BUFFERLINES	Returns the number of buffer lines as a string.
COLUMNS	Returns the column width of the current emulation (80, 132, or 180, for example).
CONNECTOR	Returns NONE, COM1, COM2, COM3, COM4; COM5, COM6, COM7, COM8, or COM9; COMBIOS port extension (where the optional extension can be null, or ETHERTERM); UBNETCI port (where port is either 1 or 2); DEVICE name (where name is the name of a device to use).
DATABITS	Returns either 4, 5, 6, 7, or 8, depending on the current value of data bits in the communications parameters.
EMULATE	Depending on the current emulation, returns one of the following values: ADDSVP60, ANSI, ATT605, ATT705, ATT4425*, IBM3270*, IBM3287*, IBM5250*, SCOANSI, TN6530-8*, TTY, Tvi925, Tvi950, VT52, VT100, VT101, VT102, VT220, VT320, VT420, Wyse60. (* only available as a client option)
FLOWCONTROL	Returns either XONXOFF, HARDWARE, or NONE, depending on the current settings of the communications parameters.
PARITY	Returns either NONE, ODD, EVEN, MARK, or SPACE, depending on the current value of parity in the communication parameters.
PHONENUMBER	Returns the current phone number used by the DIAL command. The string can contain alphanumeric characters.
RETRYDELAY	Returns the current retry delay in units of seconds.
SELECTION	Returns the line numbers selected in a session window by the SELECTION command.
SENDDelay	Returns the current send delay units (measured in sixtieths of a second).
STOPBITS	Returns either 1, 1.5, or 2, depending on the current value of stop bits in the Direct Serial parameters.
TERMFONT	Returns the current font and size of the session window in the form: fontname;fontsize.
WORDSWRAP	Returns the current column to which text is wrapped for LOGTOFILE. A value of 0 indicates that no wrapping is performed.

---

---

## SETTINGS, *continued*

---

Keyword	Description
	The following keywords return a string value, where 1 is said to be TRUE and 0 is said to be FALSE.
CARRIERDETECT	Returns 1 if carrier detect is enabled, otherwise it returns 0.
FKEYSSHOW	Returns 1 if the function keys are visible, otherwise it returns 0.
LINEWRAP	Returns 1 if incoming lines are wrapped past the end of the setting for columns, otherwise it returns 0.
LOCALECHO	Returns 1 if local echo of outgoing characters is enabled, otherwise it returns 0.
OUTGOINGCR	Returns 1 if line feeds are added to outgoing carriage returns, otherwise it returns 0.
PASSTHROUGH	Returns a string value, where 1 represents TRUE (if your printer driver supports Raw Passthrough Mode, and it is enabled), and where 0 represents FALSE (if it is not enabled).
RETRY	Returns 1 if retry is enabled, otherwise it returns 0.
SOUND	Returns 1 if sound is enabled, otherwise it returns 0.

---

### WinHandle

The optional **WinHandle** argument is an integer and identifies a particular window in DCS.

### Result

The result is a string value, depending on the particular keyword argument, as described in the keyword list above.

### Comments

If the **WinHandle** argument is not included in this function, DCS uses the window handle for the active session window.

The **ERROR** function returns TRUE if the **WinHandle** argument is not valid.

---

## SETTINGS, *continued*

---

### Example

In this example:

```
DISPLAY SETTINGS (PASSTHROUGH)
```

the value displayed in the session window is 1(one) if Raw Passthrough Mode is enabled, or 0 (zero) if not enabled.

In this example:

```
DISPLAY SETTINGS (BINARYTRANSFERPARAMS)
```

the session window displays text that describes the parameters set for the current file transfer protocol. If the current protocol is ZModem, a string similar to the following is displayed in the session window:

```
ZMODEM; PACKETSIZE 1K; TIMING LOOSE;  
ERRORCHECK AUTO; SKIP EXISTING; AUTOSTART ENABLED
```

---

# STR

---

## STR (Numeric, Precision)

The STR function returns the string value of the specified numeric.

### Arguments

#### Numeric

The Numeric argument specifies the numeric to be converted.

#### Precision

The optional **Precision** argument indicates the number of decimal places to be included in the string representation of the **Numeric** argument. It overrides the current format specified by the SET DECIMAL command. If the **Precision** argument is not included, the default precision is that specified by the SET DECIMAL command. If no SET DECIMAL has been specified, the default is zero.

### Result

The result is the string value representation of the **Numeric** argument.

### Comments

The **Precision** argument can also be a string using any valid C language format string.

### Example

This command

```
DISPLAY (0,0) STR (12345,6)
```

directs DCS to display the string 12345.000000 in the session window.

```
!measurement=ROUND(25.2314325,3)
DISPLAY (0,1) STR (!measurement,3)
```

The ROUND function returns the value 25.231, but to display this real number a string must be provided to the DISPLAY command. Therefore, the STR function is used with the optional **Precision** argument to accurately display the rounded measurement to three decimal places. Note that the **Precision** argument matches that used with the ROUND function

If the optional **Precision** argument had not been specified, and if no SET DECIMAL had been specified, the number of decimal places to be included in the string would have been 0 (zero) and DCS would only display the string 25 in the session window.

In this example:

```
DISPLAY (0,2) STR (DISKSPACE ())
```

the STR function returns the string value of DISKSPACE which is displayed in the session window.



---

# SUBSTR

---

## SUBSTR (String, Start, Length)

The SUBSTR function returns a portion of the specified string.

### Arguments

#### String

The **String** argument specifies a string from which the substring is obtained.

#### Start

The **Start** argument is an integer specifying the position of the character with which to begin the substring.

#### Length

The optional **Length** argument is an integer specifying the desired length of the substring. If the **Length** argument is not included, the substring contains all characters from the start position to the end of the string.

### Result

The result is the string value corresponding to the specified portion of the **String** argument.

### Example

In this example:

```
$sub = SUBSTR ("abcdefg", 2, 3)
```

the string variable `$sub` assumes the string value "bcd".

In this argument:

```
$sub = SUBSTR ("abcdefg", 3)
```

the string variable `$sub` assumes the string value "cdefg".

---

# SYSMETRICS

---

## SYSMETRICS ( )

The SYSMETRICS function returns the current system parameters.

### Arguments

The SYSMETRICS function takes no arguments.

### Result

The result is a string value in the following format:

```
HorizontalDisplayResolution,VerticalDisplayResolution;  
TotalDiskSpace,FreeDiskSpace;PercentMemoryInUse,GlobalFreeMemory,  
LocalFreeMemory,[WindowsMajorVersion][ProductCode]
```

### Comments

The [ProductCode] is DCSeries.

### Example

In this example:

```
$res = SYSMETRICS ( )
```

returns the result:

```
1024,768;  
1275559936,84672512;  
100,32985088,372736;[WIN95][DCSeries]
```

that shows:

- the display resolution is 1024 by 768;
- the default hard drive has about 1.3GB of disk space;
- the default hard drive has about 85MB of free disk space;
- memory is 100% in use, the global memory has about 32MB free;
- the local memory has about 372K free;
- the current version of Windows is Windows 95;
- and the product code is Dyna32.

---

# SYSTEM

---

## SYSTEM (SysNum P1, P2)

The SYSTEM function returns a string describing information about operating system-level parameters and environment variables.

 Note: This function does not apply to the IBM TN3270 emulation.

### Arguments

#### SysNum

The SysNum argument is specified by a hexadecimal value from the table below.

#### P1, P2, ...

The optional **Pn** arguments are specified by one or more integers or strings. See each entry in the table under **Example** below for specific parameters and syntax.

### Result

The result is always a string.

### Comments

The parameters listed in this electronic document are always the most up-to-date. However, please note that the parameters for this function are not universal between versions of DCS and are subject to change without notice.

 Also see: SYSTEM command

### Example

See each entry in the table for an example of syntax.

SysNum	Definition
0x0001	The function returns the file name of the currently executing script.  In the following example, the \$current_task variable will contain the name of the executing script.  <code>\$current_task = SYSTEM (0x0001)</code>
0x0950	This function returns the fully qualified path name of this script, without the file name.  In the following example, the \$scriptpath variable will contain the path of the currently executing script.  <code>\$scriptpath = SYSTEM (0x0950)</code>
0x0951	This function returns the command line of the executing application. In the following example, the \$exepath variable contains the path of the executable.  <code>\$exepath = SYSTEM (0x0951)</code>
0x8000	This function creates two temporary zero-length files and returns a string containing the file names separated by a comma.

---

## SYSTEM, *continued*

---

In the following example, the temporary file names are parsed from the result of this function and used in a dialog routine.

```
$return = system(0x8000)
parse $return $tmp1 \,' $tmp2

dialog(,,250,100) "SYSTEM(0x8000)"
  edittext(10,10,200,14) "Return Value"
  $return
  edittext(10,30,200,14) "Filename 1  " $tmp1
  edittext(10,50,200,14) "Filename 2  " $tmp2
  button (105,80,40,14) default "OK" resume
dialog end
wait resume
file delete $tmp1
file delete $tmp2
cancel
```

---

## TASKFILE

---

### TASKFILE (taskID)

Returns a string containing the path/filename of the given taskID.

#### Arguments

##### taskID

The taskID argument is an integer expression specifying the script number.

#### Result

A string containing the path/filename.

#### Comments

Use the menu option Script: Status... to see a list of running scripts.

See also the SPAWN command.

See also the TASKSTOP command.

See also the TASKLIST function.

See also the TASKNAME function.

#### Example:

```
$whoIam = TASKLIST(-1)
DIALOG
  MESSAGE $whoIam
  MESSAGE TASKFILE(NUM($whoIam))
  BUTTON "OK" RESUME
DIALOG END
WAIT RESUME
```

Displays the path/filename of the running script.

---

# TASKLIST

---

## TASKLIST (-1)

The TASKLIST function returns a list of all running scripts.

### Arguments

#### NumFlag

The optional NumFlag of -1 returns only the taskID of the current script. If no parameter is given, a list of all running taskID numbers is returned.

### Result

The result is a string composed of the taskID of all running scripts. The string is a hexadecimal representation of each script and has the generalized form of 0x\*\*\*\*, where an asterisk is a character from the set (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F). Each number, except for the last handle in the resulting string, will be followed by a comma.

Because the value returned is a string, you must use the NUM function to change the string to a numeric representation if you wish to use the result as a numeric value with other functions or commands.

### Comments

Use the menu option Script: Status... to see a list of running scripts.

See also the SPAWN command.

See also the TASKSTOP command.

See also the TASKFILE function.

See also the TASKNAME function.

### Example:

```
SPAWN "Script2"  
$whoIam = TASKLIST(-1)  
$allScripts = TASKLIST()  
DIALOG "Script1"  
  MESSAGE $whoIam  
  MESSAGE $allScripts  
  BUTTON "OK" RESUME  
DIALOG END  
WAIT RESUME
```

This runs another script (Script2) and then displays the list of running scripts. For example:

```
0x17FC,0x030C and 0x17FC  
DIALOG (5,5) "Script2"  
  BUTTON "OK" RESUME  
DIALOG END  
WAIT RESUME
```

Example Script2.dcp to test Script1

---

## TASKNAME

---

### TASKNAME (taskID)

Returns a string containing the path/filename of the given taskID.

#### Arguments

##### taskID

The taskID argument is an integer expression specifying the script number.

#### Result

A string containing the path/filename.

#### Comments

Use the menu option Script: Status... to see a list of running scripts.

See also the SPAWN command.

See also the TASKSTOP command.

See also the TASKFILE function.

See also the TASKLIST function.

#### Example:

```
$whoIam = TASKLIST(-1)
DIALOG
  MESSAGE $whoIam
  MESSAGE TASKNAME(NUM($whoIam))
  BUTTON "OK" RESUME
DIALOG END

WAIT RESUME
```

Displays the filename of the running script.

---

## TIME

---

### TIME (Seconds)

The TIME function returns the time, formatted according to the time format you have chosen via the Windows Control Panel.

#### Arguments

##### Seconds

The optional **Seconds** argument is an integer specifying a number of seconds, where zero is midnight, January 1, 1904 (see the SECONDS function). If you have not included the **Seconds** argument, DCS returns the current system time.

#### Result

The result is a string containing the time in the format specified by the time format chosen via the Windows Control Panel. For example, the time might have the following format:

Hour:Minutes:Seconds

#### Example

```
$time = TIME ()
```

If the system time is 13:45:18, \$time assumes the string value 1:45:18 PM.



---

# TIMER

---

## TIMER (Index)

The **TIMER** function returns the elapsed time since the last timer reset. DCS has access to four internal timers (accessible only through a script).

### Arguments

#### Index

The optional **Index** argument is a numeric (from zero to three) specifying the desired timer. All internal timers are initialized to the value zero (0).

### Result

The result is a string in the following format:

Hours:Minutes:Seconds

### Comments

If the **Index** argument has a value greater than 3, the **TIMER** function uses the system timer.

### Example

```
$interval = TIMER (2)
```

If 11 hours, 22 minutes and 33 seconds have elapsed since the last reset of timer two, `$interval` assumes the string value 11:22:33.

---

# TRIM

---

## TRIM (String, Pre, Post)

The TRIM function removes characters from the specified string.

### Arguments

#### String

The **String** argument specifies the string to be trimmed.

#### Pre

The optional **Pre** argument is a case-sensitive string specifying characters to discard from the front of the **String** argument. This argument specifies a character sequence, not individual characters. If you do not include the **Pre** argument, a null string is used as the default value of the **Pre** argument and nothing is trimmed from the front of the **String** argument.

#### Post

The optional **Post** argument is a case-sensitive string specifying characters to discard from the end of the **String** argument. This argument specifies a character sequence, not individual characters. If you do not include the **Post** argument, the space character is used as the default value of the **Post** argument and spaces are trimmed from the end of the **String** argument.

### Result

The result is a string value corresponding to the **String** argument with the specified **Pre** and **Post** character sequences removed.

### Example

In this example:

```
$Str = TRIM (" Houston, Texas      ")
```

the string " Houston, Texas " has one space before and five spaces after it. Since neither the **Pre** or **Post** arguments are supplied to the command, the TRIM function will remove the five trailing spaces and put the leading space and the string Houston, Texas in the string variable \$Str.

In this example:

```
$newstring = TRIM ("ABCXXXXYZ", "ABC" "XYZ")
```

the variable \$newstring assumes the string value XXX. However if the original string contained a lowercase "a" and a lowercase "y", \$newstring contains the original string (since the **Pre** and **Post** argument strings contain only uppercase characters).


---

## TYPEDLIBRARYCALL

---

### TYPEDLIBRARYCALL (LibName, ProcName, TypeString, Param1, Param2, ..., Paramn)

The TYPEDLIBRARYCALL function returns a string representing the return value from a Dynamic Link Library (DLL) function call. This function may be used to call functions from the Windows API, or from other DLLs. Note that while most DLLs have the file name extension DLL, some of the DLLs that provide the Windows API have the file name extension EXE. Refer to the Windows Software Development Kit (SDK) for more information on the use and development of Dynamic Link Libraries.

 **Note:** The calling convention of the library function being used is determined by the routine name. If the name has an underscore as the first character (“\_”), the C programming language convention is used. Otherwise, the function will be called using the standard Windows API (WINAPI) convention (sometimes referred to as the Pascal convention). It is *very important* that the calling convention be correct or unpredictable results will occur, i.e., the application may crash.

### Arguments

#### LibName

The **LibName** argument is a string specifying the complete path name to the library.

#### ProcName

The **ProcName** argument is a string specifying the name of a function or procedure in the library.

#### TypeString

The **TypeString** argument is a string listing the type of the value that the library function returns and the types of any parameters that the library function requires. The **TypeString** must match the DLL function’s return value and parameters, in order, or the call may fail. The **TypeString** argument has the following general format:

ProcRetType=ParamType1, ParamType2, ..., ParamType*n*

The only element of the string that is required in all cases is ProcRetType. If the DLL function uses any parameters, then you must add an equal sign to the string followed by the types of the parameters. Separate the parameters with a comma. The types that you may specify are any of the standard Windows SDK API types (BOOL, BYTE, DWORD, LONG, LPBYTE, LPDWORD, LPINT, LPLONG, LPSTR, LPWORD, UINT, VOID, and WORD) as described in WINDOWS.H, and C language types (char and int). All of these types can specify the type of a return value, and all except VOID can specify the type of a parameter for a library function. See the Windows SDK for more information about these types.

Param1, Param2, ..., Param*n*

The **Param1, Param2, ..., Param*n*** arguments will be type cast according to the **TypeString**, and then used as the parameters to the DLL function call. These parameters have a one-to-one correspondence to the **ParamType** part of the **TypeString** argument. Also, the parameters are separated by commas.

---

## TYPEDLIBRARYCALL, *continued*

---

### Result

The TYPEDLIBRARYCALL function returns a string representing the return value from a Dynamic Link Library (DLL) function call.

### Examples

```
$Result = TypedLibraryCall ("user32.dll", "MessageBeep, \  
"Bool=UINT, 0)
```

This example calls the Windows API function MessageBeep, which is in USER32.dll.

```
%Length = Num (TypedLibraryCall ("user32.dll", "_ws-  
printf", \  
"int=LPSTR, LPSTR, LONG", $Output, "the value is  
%ld", %value))
```

This example uses the 'C' call convention to call the Windows API function `wsprintf`, which is in `USER32.dll`. The API function returns a C language integer value, so the `TypeString` indicates an "int" return type. The API function expects two long pointers to strings as parameters, and also a `LONG` in this case, so the `TypeString` indicates two "LPSTR" parameters and one "LONG", separated by commas. If `%value` is 365, the API will fill the `$Output` script language variable with "the value is 365".

---

# UPPER

---

## UPPER (String)

The UPPER function returns the specified string with the characters converted to their uppercase equivalents.

### Arguments

#### String

The String argument specifies the string to be converted to uppercase.

### Result

The result is a string value corresponding to the String argument, with all characters converted to uppercase.

### Example

In this example:

```
$city = UPPER ("Houston, Texas")
```

the variable `$city` assumes the string value "HOUSTON, TEXAS".

---

## USERID

---

### USERID ( )

The USERID function returns the current value of the UserID system variable.

 **Note:** This function does not apply to the IBM TN3270 emulation.

### Result

The result is the current value of the UserID system variable. The string can contain alphanumeric characters.

### Comments

Compare with the SET USERID command.

### Example

This example:

```
WAIT STRING "UserName:"  
SEND UserID ( )
```

assumes the UserID system variable has been given a value using the SET USERID command. When the host sends the user name prompt and the string UserName: is displayed in the session window, DCS sends the string contained in the UserID system variable to the host.

---

## VERSION

---

### VERSION ( )

The VERSION function returns the current DCS version number.

#### Arguments

The VERSION function takes no arguments.

#### Result

The result is a string value containing the release version number of the currently running DCS program.

#### Example

In this example:

```
dialog
message "Current version is: " | version ()
button "OK" resume
dialog end
wait resume
```

the current version of DCS is displayed in a dialog box.

---

# VISIBLE

---

## VISIBLE (WinHandle)

The VISIBLE function indicates whether a particular window is visible (or not hidden with the WINDOW HIDE command).

### Arguments

#### WinHandle

The WinHandle argument is an integer expression specifying a particular window. The DCS application window has the handle number zero.

### Result

The result is a Boolean value indicating if the window is visible. If it evaluates to TRUE, the window is visible. If the function evaluates to FALSE, the window is hidden.

### Comments

A window is always considered visible, even if completely obscured by other windows, unless hidden by the WINDOW HIDE command.

If the WinHandle argument is not included in this function, DCS uses the window handle for the active child window.

The ERROR function returns TRUE if the WinHandle argument is not valid.

### Example

This script example:

```
IF VISIBLE (%wnd)
  $where = Position (%wnd)
```

employs the VISIBLE function to determine if a window designated by the variable %wnd is visible. If the window is visible, the result of the POSITION function is placed in the string variable \$where.



---

# WINDOW

---

## WINDOW (WinHandle)

The WINDOW function indicates whether a particular window is a child of the main DCS window.

### Arguments

#### WinHandle

The WinHandle argument is an integer expression specifying a particular window handle.

### Result

The result is a Boolean value, where TRUE indicates that the specified window is a child window of the main DCS window, and FALSE indicates that it is not.

### Comments

If the WinHandle argument is not included in this function, DCS uses the window handle for the active child window.

The ERROR function returns TRUE if the WinHandle argument is not valid.

### Example

In this script example:

```
IF WINDOW (%wnd)
WINDOW MINIMIZE %wnd
```

if the window designated by the variable %wnd is a child window of the DCS application window, it is minimized. The icon for the minimized window will be located near the bottom of the DCS application window.

---

# WINDOWHND

---

## WINDOWHND (WinName)

The WINDOWHND function returns the handle of the window whose title is specified by the WinName argument.

### Arguments

#### WinName

The WinName argument is a string expression specifying a particular window title. This is the title as displayed in the window title bar, to which DCS appends the file type extension (.ses, .dct, .dcm). This argument can be input with or without the file type extension.

### Result

The result is an integer, which is the window handle of the window.

### Comments

The ERROR function returns TRUE if the WinName argument is not valid.

If there are multiple windows open with the same name, the window handle for the first one found will be returned. If there are windows open with the same name but of a different type (e.g., bbs . ses and bbs . dcm), the window handle for the first one found will be returned.

### Example

In this example:

```
%hnd = WINDOWHND ("tr1")
```

the window handle of the window titled "tr1" is assigned to the variable %hnd.

---

## WINDOWNAME

---

### WINDOWNAME (WinHandle)

The WINDOWNAME function returns the title of the window specified by the WinHandle argument.

#### Arguments

##### WinHandle

The WinHandle argument is an integer and identifies a particular window in DCS.

#### Result

The result is a string, which is the name of the window.

#### Comments

The window name is the fully qualified path name of session, script or memo files, and null in the case of an untitled window.

The ERROR function returns TRUE if the WinHandle argument is not valid.

#### Example

In this example:

```
$name = WINDOWNAME (0xB4A)
```

the path name of the window with the window handle "0xB4A" is assigned to the variable \$name.

---

# WNDCLASS

---

## WNDCLASS (WinHandle)

The WNDCLASS function returns the DCS window type of a specified window.

### Arguments

#### WinHandle

The WinHandle argument is an integer expression specifying a particular window.

### Result

The result is an integer value that corresponds to the window type. If ERROR is set to TRUE, the result is -1 (negative one).

Value	Window Type
0	DCS main window
1	Session window
3	Script window
4	Memo window

### Comments

The WinHandle argument must be included when using this function.

The ERROR function returns TRUE if the WinHandle argument is not valid.

### Example

This script example:

```
%WinType = WNDCLASS (%WND)
```

assigns an integer to the %WinType variable. This integer represents the window type.

---

## WNDFILE

---

### WNDFILE (WinHandle)

The `WNDFILE` function retrieves the disk file name associated with the specified window.

#### Arguments

##### `WinHandle`

The `WinHandle` argument is an integer expression specifying a particular window.

#### Result

The result is a string value that is the disk file name of the window.

#### Comments

The `ERROR` function returns `TRUE` if the `WinHandle` argument is not valid.

---

# WNDTITLE

---

## WNDTITLE (WinHandle)

The WNDTITLE function retrieves the window title.

### Arguments

#### WinHandle

The WinHandle argument is an integer expression specifying a particular window.

### Result

The result is a string value that is the window title.

### Comments

The ERROR function returns TRUE if the WinHandle argument is not valid.

Compare with the SET WINDOWTITLE command.

### Example

This script segment:

```
$TITLE = WNDTITLE (%WND)
```

assigns the title of the window designated by the variable %WND to the variable \$TITLE.

---

## ZOOMED

---

### ZOOMED (WinHandle)

The ZOOMED function indicates whether a particular window is maximized.

#### Arguments

##### WinHandle

The **WinHandle** argument is an integer expression specifying a particular window. Use a WinHandle of 0 (zero) to specify the main application window.

#### Result

The result is a Boolean value, where TRUE indicates that the specified window is maximized, and FALSE indicates that it is not.

#### Comments

If the WinHandle argument is not included in this function, DCS uses the window handle for the active child window.

The ERROR function returns TRUE if the WinHandle argument is not valid.

#### Example

In this example:

```
IF ZOOMED (%WND)
WINDOW RESTORE (%WND)
```

if the window is zoomed, the ZOOMED function evaluates to TRUE, and the window is restored to its previous state.

# 3

---

## *Commands*

DCS



---

## Commands in Alphabetical Order

---

 **Note:** Commands flagged with an asterisk do not apply to the IBM TN3270 emulation.

### —A—

APPCONFIG	Specifies parameter used to configure the DCS application
ARGUMENTS	Specifies local variables to be created by a child routine

### —B—

BEEP*	Sounds the default system .wav file a number of times
BEGIN	Specifies the beginning of a command block
BREAK*	Sends a break signal to the remote system

### —C—

CANCEL	Cancels script execution
CLEAR	Clears the screen and the history buffer
COLLECT*	Stores incoming characters in the specified string
COMPILE	Compiles the specified DCS script
CONCAT	Concatenates two or more strings
CONNCONFIG	Sets a connector parameter for the active session
CONNECT	Establishes a connection with a remote system
CONTINUE	Conditionally branches execution from a WHILE loop
CREATE DIRECTORY*	Creates the specified directory

### —D—

(DDE) ACCESS	Initiates a DDE conversation
(DDE) ACCESS CANCEL	Terminates the active DDE conversation
(DDE) INSTRUCT	Sends a string of commands to the DDE server
(DDE) POKE	Sends a single data item to the DDE server
(DDE) REQUEST	Requests a single data item from the DDE server
(DDE) TABLE REPLY	Sends a table to a DDE client in response to an ADVISE or REQUEST
(DDE) TABLE REQUEST	Requests data in the specified format from the DDE server
(DDE) TABLE SEND	Sends the specified table to the DDE server
(DDE) WAIT SIGNAL	Pauses execution until the next DDE REQUEST is received
(DDE) WHEN ADVISE	Activates when a DDE ADVISE request is received
(DDE) WHEN EXECUTE	Activates when a DDE EXECUTE request is received
(DDE) WHEN INITIATE	Activates when a DDE INITIATE request is received
(DDE) WHEN POKE	Activates when a DDE POKE request is received

---

## Commands in Alphabetical Order, *continued*

---

(DDE) WHEN REQUEST	Activates when a DDE REQUEST is received
(DDE) WHEN TERMINATE	Activates when the current DDE conversation is terminated
DEBUG	Writes all script commands to the specified file as they execute
DECREMENT	Decreases the value of an integer variable by one
DIAL*	Dials the specified number using the modem
DIALOG	Creates a dialog box
(DIALOG) BUTTON	Creates a button control in a dialog box
DIALOG CANCEL	Removes the active dialog box from the screen
(DIALOG) CHECKBOX	Creates a check box control in a dialog box
DIALOG CONTROL*	Updates the attributes of a previously defined dialog control
(DIALOG) DIMENSION	Specifies the size and position of a control in a dialog box
(DIALOG) EDITTEXT	Creates an edit text control in a dialog box
(DIALOG) GROUPBOX	Creates a group box control in a dialog box
(DIALOG) ICON	Displays the specified icon in a dialog box
(DIALOG) ICONBUTTON	Creates an icon button control in a dialog box
(DIALOG) LISTBOX	Creates a list box control in a dialog box
(DIALOG) MESSAGE	Displays a text message in a dialog box
(DIALOG) NEWLINE	Positions the next dialog control on the next line of the dialog box
(DIALOG) PICTURE	Displays the specified picture in a dialog box
(DIALOG) RADIOBUTTON	Creates a radio button control in a dialog box
(DIALOG) RADIOGROUP	Defines a group of radio button controls in a dialog box
DIALOG UPDATE	Updates a previously defined dialog box
DISCONNECT	Terminates a connection with a remote system
DISPLAY	Displays a string in the session window (does not send the string to a remote system)
DISPLAYCONFIG	Changes the visual characters of a session window
DROPDTR*	Holds the DTR line low for the current serial port
<b>—E—</b>	
EDIT COPY	Copies the specified string to the clipboard
EDIT COPYSPECIAL*	Implements the <b>Copy Special</b> selection on the <b>Edit</b> menu for the session window in the script language
EDIT CUT	Cuts the current selection in the active edit window to the clipboard

---

## Commands in Alphabetical Order, *continued*

---

EDIT FIND	Searches the active edit window for the specified string
EDIT GOTO	Positions the cursor at the beginning of the specified line
EDIT PASTE	Pastes text from the clipboard in the active edit window
EDIT REPLACE	Searches an edit window for a string and replaces it with another string
EMULCONFIG	Sets an emulator parameter for the active session
END	Specifies the end of a command block
EXECUTE	Starts a script and does not return to the calling script
<b>—F—</b>	
FILE CLOSE*	Terminates the text transfer process initiated by a LOGTO-FILE command
FILE COMPRESS	Converts a binary file to a seven data-bit ASCII format
FILE COPY	Copies the contents of the source file to the destination file
FILE CREATENAME	Prompts you for the name of a file to create
FILE DECOMPRESS	Converts a file from a seven data-bit ASCII format to a binary format
FILE DECRYPT	Decrypts the specified encrypted file to be readable
FILE DELETE	Deletes the specified file or directory
FILE ENCRYPT	Encrypts the specified file to be unreadable
FILE OPENNAME	Prompts you for the name of a file to open
FILE PAUSE*	Suspends the current text transfer initiated by a LOGTO-FILE command
FILE RECEIVE BINARY	Prepares DCS to receive the specified binary file
FILE RENAME	Renames the specified file
FILE RESUME*	Resumes the text transfer suspended by the FILE PAUSE command
FILE SEND BINARY	Sends the specified binary file to the remote system
<b>—G—</b>	
GENERALCONFIG	Specifies parameters for general session configuration options.
GOTO	Branches execution to the specified target
<b>—H—</b>	
HANGUP*	Disconnects the telephone line on the modem

---

## Commands in Alphabetical Order, *continued*

---

### —I—

IF	Conditionally executes a command block
INCREMENT	Increases the value of an integer variable by one

### —K—

KERMIT COPY*	Copies a file on a remote host to another place on the remote host
KERMIT DIRECTORY*	Lists the contents of a directory on a remote system
KERMIT ERASE*	Deletes a file from directory on a remote system
KERMIT FINISH*	Sends a Kermit finish packet to the remote Kermit server
KERMIT FREESPACE*	Displays the number of free bytes on the remote system
KERMIT HELP*	Displays a summary of a topic on the remote system's operation
KERMIT LOGOUT*	Sends a Kermit logout packet to the remote Kermit server
KERMIT MESSAGE*	Sends a short message to an account on the remote system
KERMIT NEWDIRECTORY*	Changes the current working directory on a remote system
KERMIT RENAME*	Changes the name of a file on a remote system
KERMIT TYPE*	Displays the contents of a file from the remote system
KERMIT WHO*	Issues the KERMIT WHO command to the remote system
KEY	Remaps a given key combination to the specified definition
KEYBOARD	Locks and unlocks the keyboard
KEYMAP LOAD	Loads the specified keymap
KEYMAP RESET	Determines whether or not keystrokes are transmitted to the local or remote system
KEYMAP SAVE	Saves the specified keymap

### —L—

LAUNCH	Starts another application
LEAVE	Branches execution from a command block
LEVEL	Specifies a given function key level
LIBRARY CALL	Branches execution to the specified DLL procedure
LIBRARY LOAD	Loads the specified DLL into memory
LIBRARY UNLOAD	Removes the specified DLL from memory
LINENUMBERS	Inserts line numbers into a compiled script
LOAD	Loads a session file
LOGTOFILE	Saves incoming data to a specified file.

---

## Commands in Alphabetical Order, *continued*

---

### —M—

MENU	Creates or modifies a menu or menu item
MENU CANCEL	Removes a script menu
(MENU) DELETE ITEM	Removes a menu item from a popup menu
(MENU) DELETE POPUP	Removes a popup menu from the menu bar
(MENU) INSERT ITEM	Inserts a menu item on a popup menu
(MENU) INSERT POPUP	Inserts a popup menu on the menu bar
(MENU) ITEM	Adds a menu item to a popup menu
(MENU) POPUP	Adds a new popup menu to the menu bar
(MENU) SEPARATOR	Adds a menu item separator to a popup menu
MENU UPDATE	Updates a previously defined menu option, or group of options

### —N—

NOSHOW	Suspends script command trace display
--------	---------------------------------------

### —P—

PARSE	Locates a substring in another string, and stores preceding and succeeding characters from the string
PERFORM	Branches execution to the specified target
PRINT CANCEL	Terminates active print jobs
PRINT CLOSE	Closes the open print channel and terminates any active print jobs
PRINT FILE	Prints a file
PRINT FONT	Changes the active print font
PRINT NEWLINE	Sends a carriage return and line feed to the printer
PRINT NEWPAGE	Sends a form feed to the printer
PRINT OPEN	Opens a print channel
PRINT STRING	Prints the specified string
PRINT STYLE	Changes the active print character characteristics
PRINT TABS	Specifies the print tab width
PRINT TERMINAL*	Sends the incoming session window data to the printer

### —Q—

QUIT	Cancels all script execution and terminates the DCS session
------	---

### —R—

RECORD FORMAT	Defines a virtual record template for session window data fields (used with the RECORD SCAN command)
---------------	--

---

## Commands in Alphabetical Order, *continued*

---

RECORD READ	Reads data from a structured or text table into its record buffer
RECORD SCAN	Retrieves data from the session window into a table defined by the RECORD FORMAT command
RECORD WRITE	Writes the contents of the record buffer to a structured or text table
REMOVE DIRECTORY*	Deletes the specified directory
RESTART	Branches execution to the first line of the executing script
RESUME	Resumes script execution past the current WAIT
RETURN	Resumes script execution past the current PERFORM
<b>—S—</b>	
SAVE	Saves the current settings file
SCREEN	Specifies the size and position of the session window (shows and hides terminal; turns on and off updating)
SCROLL DOWN	Scroll down a specified number of lines in session history buffer
SCROLL LEFT	Scroll left a specified number of lines in session history buffer
SCROLL RIGHT	Scroll right a specified number of lines in session history buffer
SCROLL UP	Scroll up a specified number of lines in session history buffer
SELECTION	Selects a block in the session window
SELECTION APPEND	Appends the current selection to the specified file
SELECTION BUFFER	Selects the entire contents of the history buffer
SELECTION PRINT	Prints the current selection
SELECTION SAVE	Saves the current selection to the specified file
SELECTION SEND	Sends the current selection to the remote system
SEND	Sends the specified string to the remote system
SEENDBREAK*	Sends a break signal to the remote system
SET	Assigns a value to the specified variable
SET APPTITLE	Displays the specified string in the application's title bar
SET ATTRIBUTES	Modifies the file attributes of the specified file
SET AUTOSCROLLTOCURSOR	Determines whether emulation scrolling follows the cursor
SET AUTOSIZE*	Determines the status of the <b>Automatically Size</b> check box on the <b>Display</b> tab of the <b>Session Properties</b> dialog
SET BACKSPACEDESTRUCTIVE*	Determines whether the backspace key is destructive

---

## Commands in Alphabetical Order, *continued*

---

SET BACKSPACEKEY*	Specifies the character sent when the [Backspace] key is pressed
SET BAUDRATE*	Changes the baud rate of the selected communications port
SET BINARYTRANSFERPARAMS*	Changes the settings for a binary transfer protocol
SET BINARYTRANSFERS	Sets the current binary transfer protocol or host environment
SET BUFFERLINES	Specifies the number of lines in the session window and its history buffer
SET CARRIERDETECT*	Sets the carrier detect flag
SET COLUMNS	Specifies the number of columns in the session window
SET CONNECTION*	Specifies a communications connector, including those from third parties
SET CONNECTMESSAGE*	Changes the connection message from the Phone Book
SET CONNECTRESULT*	Changes the value of the ConnectResult variable
SET CURSOR	Sets the shape of the cursor and turns display on and off
SET DATABITS*	Sets the number of data bits to be used during transmission
SET DDETIMEOUT	Sets the timeout period for DDE commands
SET DECIMAL	Specifies the number of decimal places for string representations of real numbers
SET DEFAULTSESSIONHANDLE	Specifies a handle for the default session window
SET DIRECTORY	Sets the directory location for a type of file
SET EMULATION*	Loads the specified terminal emulation DLL
SET FKEYSSHOW	Sets the function keys to appear when file opened
SET FLOWCONTROL*	Specifies the desired flow control method
SET KEEPPRINTCHANNELOPEN	Relinquishes Windows print channel
SET LOCALECHO*	Shows and hides local keystrokes
SET NETID*	Changes the characters of the NetID system variable
SET OUTGOINGCR*	Appends line feed characters to outgoing carriage returns
SET PARITY*	Specifies the type of parity to be used during data transmission
SET PASSTHROUGH*	Sets the Raw Passthrough Mode for a printer
SET PASSWORD*	Changes the characters of the Password system variable
SET PHONENUMBER*	Specifies the default phone number for the DIAL command
SET RESULT	Assigns a value to the RESULT string
SET RETRY*	Specifies whether to continue redialing until connection
SET RETRYDELAY*	Sets the delay time between dial attempts
SET SENDDELAY	Specifies the amount of time to wait between transmitting characters

---

## Commands in Alphabetical Order, *continued*

---

SET SIGNAL*	Allows the system bell to sound when a connection is made
SET SOUND	Controls whether warning bells from the host are enabled
SET STOPBITS*	Sets the number of stop bits used during serial transmissions
SET TERMCLOSE	Allows or disallows you to close the session window
SET TERMFONT	Specifies the font displayed in the session window
SET USERID*	Changes the characters of the UserID system variable
SET WILDCARD*	Sets the characters representing any arbitrary characters
SET WINDOWTITLE	Displays the specified string in the active window's title bar
SET WORDWRAP*	Specifies the column at which to wrap incoming text
SET XCLOCK	Specifies how long to wait for an XCLOCK host message to clear before generating an execution error during a send
SET XSYSTEM	Specifies how long to wait for an XSYSTEM host message to clear before generating an execution error during a send
SETTINGS	Allows you to modify options on a <b>Session Properties</b> tab
SHOW	Resumes command audit initiated by DEBUG command
SPAWN	Starts an independent script without interrupting current script
SWITCH	Branches execution to multiple command blocks
SYSTEM*	Performs a specialized system operation
<b>—T—</b>	
TABLE CLEAR	Clears any existing data from the specified table
TABLE CLOSE	Closes the specified table
TABLE COPY	Copies the contents of a given table to the specified table
TABLE DEFINE	Prepares a table for random access file operations
TABLE LOAD	Imports data from a file or the clipboard to the specified table
TABLE SAVE	Exports the contents of the specified table to a file or the clipboard
TABLE SORT	Performs a sort on the specified table using the given criteria
TASKERROR	Calls DCS's internal execution error handling routine
TIMER RESET	Resets the specified timer
TITLE	Assigns a title, displayed in the application title bar, to a task file
TOOLBARHIDE	Closes the specified toolbar.
TOOLBARSHOW	Displays the specified toolbar.
TRANSFERS	Specifies parameters for file transfers with a 3270 command processor



---

## Commands in Alphabetical Order, *continued*

---

### —W—

WAIT CHAR*	Pauses script execution until specified character is received
WAIT CLOSE*	Pauses execution while LOGTOFILE accesses a file
WAIT DELAY	Pauses execution until after a specified amount of time or RESUME
WAIT ECHO	Pauses execution until DCS receives a character
WAIT EDIT	Opens a memo window and pauses execution
WAIT PROMPT*	Pauses execution until a number of characters are received
WAIT QUIET	Pauses execution until an amount of time elapses without character transmission
WAIT RESUME	Pauses execution until a RESUME command is executed
WAIT SCREEN	Pauses execution until remote host changes the data in the terminal window
WAIT STRING*	Pauses execution until the specified string is received
WAIT UNTIL	Pauses execution until the specified time of day
WHEN CANCEL	Cancels the specified WHEN commands
WHEN COLLECT	Activates storing screen data to a string
WHEN DISCONNECT	Activates when the communications connection is terminated
WHEN ECHO	Activates when any character is received
WHEN ERROR	Activates when an execution error is encountered
WHEN INPUT	Activates when a keystroke-generated character is transmitted
WHEN QUIET	Activates after time elapses without character transmission
WHEN SCREEN	Activates when the specified screen region is modified
WHEN STRING*	Activates when the specified string is received
WHEN TIMER	Activates after the specified amount of time has elapsed
WHEN WINDOW*	Filters out Windows messages to child windows
WHILE	Conditionally continuously executes a logical command
WINDOW ACTIVATE	Brings the specified window into focus
WINDOW ARRANGE	Tiles all open document windows
WINDOW CLOSE	Closes the specified window
WINDOW DEFAULT	Passes a Windows message intercepted by the WHEN WINDOW command to the default handler for a window
WINDOW HIDE	Hides the application window and its child window
WINDOW MAXIMIZE	Maximizes the specified window

---

## Commands in Alphabetical Order, *continued*

---

WINDOW MESSAGE	Sends a Windows message to the specified window
WINDOW MINIMIZE	Sets DCS to run as an icon
WINDOW MOVE	Moves and resizes the specified window
WINDOW OPEN	Opens a new window, or one containing an existing file
WINDOW RESTORE	Restores the DCS application window to its prior state
WINDOW STACK	Stacks all open windows in cascade fashion
WINDOW UNHIDE	Displays the application window and its child window


### —X—

XFERCONFIG	Sets a file transfer parameter for the active session
------------	---

---

## Commands by Category

---

 **Note:** Commands flagged with an asterisk do not apply to the IBM TN3270 emulation.

### Assignment Commands

SET	Assigns a value to the specified variable
SET RESULT	Assigns a value to a RESULT string

### Branching, Subroutine, and Conditional Commands

ARGUMENTS	Specifies local variables to be created by a child routine
BEGIN	Specifies the beginning of a command block
CONTINUE	Conditionally branches execution from a WHILE loop
(DDE) WAIT SIGNAL	Pauses execution until the next DDE REQUEST is received
(DDE) WHEN ADVISE	Activates when a DDE ADVISE request is received
(DDE) WHEN EXECUTE	Activates when a DDE EXECUTE request is received
(DDE) WHEN INITIATE	Activates when a DDE INITIATE request is received
(DDE) WHEN POKE	Activates when a DDE POKE request is received
(DDE) WHEN REQUEST	Activates when a DDE REQUEST is received
(DDE) WHEN TERMINATE	Activates when the current DDE conversation is terminated
END	Specifies the end of a command block
EXECUTE	Starts a script and does not return to the calling script
FILE PAUSE*	Suspends the current text transfer
GOTO	Branches execution to the specified target
IF	Conditionally executes a command block
LAUNCH	Starts another application
LEAVE	Branches execution from a command block
LIBRARY CALL	Branches execution to the specified DLL procedure
PERFORM	Branches execution to the specified target
RESTART	Branches execution to the first line of the executing script
RESUME	Resumes script execution past the current WAIT
RETURN	Resumes script execution past the current PERFORM
SPAWN	Starts an independent script without interrupting current script
SWITCH	Branches execution to multiple command blocks
WAIT CHAR*	Pauses script execution until specified character is received
WAIT CLOSE*	Pauses execution while LOGTOFILE accesses a file
WAIT DELAY	Pauses execution until after a specified amount of time or RESUME
WAIT ECHO	Pauses execution until a character is received

---

## Commands by Category, *continued*

---

WAIT EDIT	Opens a memo window and pauses execution
WAIT PROMPT*	Pauses execution until a number of characters are received
WAIT QUIET	Pauses execution until time elapses without character transmission
WAIT RESUME	Pauses execution until a RESUME command is executed
WAIT SCREEN	Pauses execution until remote host changes the data in the terminal window
WAIT STRING*	Pauses execution until the specified string is received
WAIT UNTIL	Pauses execution until the specified time of day
WHEN CANCEL	Cancels the specified WHEN commands
WHEN COLLECT	Activates storing screen data to a string
WHEN DISCONNECT	Activates when the communications connection is terminated
WHEN ECHO	Activates when any character is received
WHEN ERROR	Activates when an execution error is encountered
WHEN INPUT	Activates when a keystroke-generated character is transmitted
WHEN QUIET	Activates after time elapses without character transmission
WHEN SCREEN	Activates when the specified screen region is modified
WHEN STRING*	Activates when the specified string is received
WHEN TIMER	Activates after the specified amount of time has elapsed
WHEN WINDOW*	Filters out Windows messages to child windows
WHILE	Conditionally continuously executes a logical command

### Command Block Commands

BEGIN	Specifies the beginning of a command block
END	Specifies the end of a command block

### Configuration Commands

APPCONFIG	Specifies parameter used to configure the DCS application.
CONNCONFIG	Sets a connector parameter for the active session
DISPLAYCONFIG	Changes the visual characters of a session window
EMULCONFIG	Sets an emulator parameter for the active session
GENERALCONFIG	Specifies parameters for general session configuration options
SET BINARYTRANSFERS	Sets the current binary transfer protocol or host environment

---

## Commands by Category, *continued*

---

SET CONNECTION*	Specifies a communications connector, including from third parties
SET EMULATION*	Loads the specified terminal emulation DLL
XFERCONFIG	Sets a file transfer parameter for a session

### Conversion Commands

FILE COMPRESS	Converts a binary file to a seven data-bit ASCII format
---------------	---

### Data Transfer Commands

COLLECT*	Stores incoming characters in the specified string
FILE CLOSE*	Terminates the current text transfer being performed
FILE COMPRESS	Converts a binary file to a seven data-bit ASCII format
FILE DECOMPRESS	Converts a file from a seven data-bit ASCII format to a binary format
FILE PAUSE*	Suspends the current text transfer being performed
FILE RECEIVE BINARY	Prepares DCS to receive the specified binary file
FILE RESUME*	Resumes the text transfer suspended by the FILE PAUSE command
FILE SEND BINARY	Sends the specified binary file to the remote system
KERMIT COPY*	Copies a file on a remote host to another place on the remote host
KERMIT DIRECTORY*	Lists the contents of a directory on a remote system
KERMIT ERASE*	Deletes a file from a directory on a remote system
KERMIT FINISH*	Sends a Kermit finish packet to the remote Kermit server
KERMIT FREESPACE*	Displays the number of free bytes on the remote system
KERMIT HELP*	Displays a summary of a topic on the remote system's operation
KERMIT LOGOUT*	Sends a Kermit logout packet to the remote Kermit server
KERMIT MESSAGE*	Sends a short message to an account on the remote system
KERMIT NEWDIRECTORY*	Changes the current working directory on a remote system
KERMIT RENAME*	Changes the name of a file on a remote system
KERMIT TYPE*	Displays the contents of a file from the remote system
KERMIT WHO*	Issues the KERMIT WHO command to the remote system
LOGTOFILE	Saves incoming data to a specified file.
SELECTION SEND	Sends the current selection to the remote system
SEND	Sends the specified string to the remote system
SET BINARYTRANSFERPARAMS	Changes the settings for a binary transfer protocol

---

## Commands by Category, *continued*

---

SET BINARYTRANSFERS	Sets the current binary transfer protocol or host environment
SET DATABITS*	Sets the number of data bits to be used during serial transmission
SET OUTGOINGCR*	Appends line feed characters to outgoing carriage returns
SET PARITY*	Specifies the type of parity to use during data transmission
SET SENDDELAY	Specifies the amount of time to wait between transmitting characters
SET STOPBITS*	Specifies the number of stop bits to be used during a connection
SET XCLOCK	Specifies how long to wait for an XCLOCK host message to clear before generating an execution error during a send
SET XSYSTEM	Specifies how long to wait for an XSYSTEM host message to clear before generating an execution error during a send
TRANSFERS	Specifies parameters for file transfers with a 3270 command processor
WAIT CLOSE*	Pauses a script while the LOGTOFILE command access a file

### Dialog Commands

DIALOG	Creates a dialog box
(DIALOG) BUTTON	Creates a button control in a dialog box
DIALOG CANCEL	Removes the active dialog box from the screen
(DIALOG) CHECKBOX	Creates a check box control in a dialog box
DIALOG CONTROL*	Updates the attributes of a previously defined dialog control
(DIALOG) DIMENSION	Specifies the size and position of a control in a dialog box
(DIALOG) EDITTEXT	Creates an edit text control in a dialog box
(DIALOG) GROUPBOX	Creates a group box control in a dialog box
(DIALOG) ICON	Displays the specified icon in a dialog box
(DIALOG) ICONBUTTON	Creates an icon button control in a dialog box
(DIALOG) LISTBOX	Creates a list box control in a dialog box
(DIALOG) MESSAGE	Displays a text message in a dialog box
(DIALOG) NEWLINE	Positions the next dialog control on the next line of the dialog box
(DIALOG) PICTURE	Displays the specified picture in a dialog box
(DIALOG) RADIOBUTTON	Creates a radio button control in a dialog box
(DIALOG) RADIOGROUP	Defines a group of radio button controls in a dialog box
DIALOG UPDATE	Updates a previously defined dialog box

---

## Commands by Category, *continued*

---

### Dynamic Data Exchange

(DDE) ACCESS	Initiates a DDE conversation
(DDE) ACCESS CANCEL	Terminates the active DDE conversation
(DDE) INSTRUCT	Sends a string of commands to the DDE server
(DDE) POKE	Sends a single data item to the DDE server
(DDE) REQUEST	Requests a single data item from the DDE server
(DDE) TABLE REPLY	Sends a table to a DDE client in response to an ADVISE or REQUEST
(DDE) TABLE REQUEST	Requests data in the specified format from the DDE server
(DDE) TABLE SEND	Sends the specified table to the DDE server
(DDE) WAIT SIGNAL	Pauses a script until a client receives another DDE request
(DDE) WHEN ADVISE	Activates when a DDE ADVISE request is received
(DDE) WHEN EXECUTE	Activates when a DDE EXECUTE request is received
(DDE) WHEN INITIATE	Activates when a DDE INITIATE request is received
(DDE) WHEN POKE	Activates when a DDE POKE request is received
(DDE) WHEN REQUEST	Activates when a DDE REQUEST is received
(DDE) WHEN TERMINATE	Activates when the current DDE conversation is terminated
LAUNCH	Starts another application
SET DDETIMEOUT	Sets the timeout period for DDE commands

### Dynamic Link Library Commands

LIBRARY CALL	Branches execution to the specified DLL procedure
LIBRARY LOAD	Loads the specified DLL into memory
LIBRARY UNLOAD	Removes the specified DLL from memory
SET EMULATION*	Loads the specified terminal emulation DLL

### Edit Commands

EDIT COPY	Copies the specified string to the clipboard
EDIT COPYSPECIAL*	Implements the <b>Copy Special</b> selection on the <b>Edit</b> menu for the session window in the script language
EDIT CUT	Cuts the current selection in the active edit window to the clipboard
EDIT FIND	Searches the active edit window for the specified string
EDIT GOTO	Positions the cursor at the beginning of the specified line
EDIT PASTE	Pastes text from the clipboard in the active edit window
EDIT REPLACE	Searches an edit window for a string and replaces it with another string

---

## Commands by Category, *continued*

---

PARSE	Locates a substring in another string, and stores preceding and succeeding characters from the string
SELECTION	Selects a block in the session window
SELECTION APPEND	Appends the current selection to the specified file
SELECTION BUFFER	Selects the entire contents of the history buffer
SET WILDCARD*	Sets the characters representing any arbitrary characters
TABLE SORT	Performs a sort on the specified table using the given criteria
WAIT EDIT	Opens a memo window and pauses execution until it is closed

### File Commands

CREATE DIRECTORY*	Creates the specified directory
EDIT COPYSPECIAL*	Implements the <b>Copy Special</b> selection on the <b>Edit</b> menu
FILE CLOSE*	Terminates a text transfer performed by a LOGTOFILE or FILE VIEW TEXT command
FILE COMPRESS	Converts a binary file to a seven data-bit ASCII format
FILE COPY	Copies the contents of the source file to the destination file
FILE CREATENAME	Prompts you for the name of a file to create
FILE DECOMPRESS	Converts a file from a seven data-bit ASCII format to a binary format
FILE DECRYPT	Decrypts the specified encrypted file to be readable
FILE DELETE	Deletes the specified file or directory
FILE ENCRYPT	Encrypts the specified file to be unreadable
FILE OPENNAME	Prompts you for the name of a file to open
FILE PAUSE*	Suspends a text transfer command until FILE RESUME executes
FILE RECEIVE BINARY	Prepares DCS to receive a binary file
FILE RENAME	Renames the specified file
FILE RESUME*	Resumes the text transfer suspended by the FILE PAUSE command
FILE SEND BINARY	Sends the specified binary file to the remote system
LOAD	Loads a session file
LOGTOFILE	Saves incoming data to a specified file.
PRINT FILE	Prints a file
REMOVE DIRECTORY*	Deletes the specified directory
SAVE	Saves the current settings file
SELECTION APPEND	Appends the current selection to the specified file



---

## Commands by Category, *continued*

---

SELECTION SAVE	Saves the current selection to the specified file
SET ATTRIBUTES	Modifies the file attributes of the specified file
SET DIRECTORY	Sets the directory location for a type of file
TABLE DEFINE	Prepares a table for random access file operations
TABLE LOAD	Imports data from a file or the clipboard to the specified table
TABLE SAVE	Exports the contents of the specified table to a file or the clipboard
WAIT CLOSE*	Pauses execution until LOGTOFILE closes its file
WAIT EDIT	Opens a memo window and pauses execution
WINDOW CLOSE	Closes the specified window and prompts you to save any changes
WINDOW OPEN	Opens a new window, or one containing an existing file

### Math Commands

DECREMENT	Decreases the value of an integer variable by one
INCREMENT	Increases the value of an integer variable by one

### Menu & Toolbar Commands

MENU	Creates or modifies a menu or menu item
MENU CANCEL	Removes a script menu
MENU DELETE ITEM	Removes a menu item from a popup menu
MENU DELETE POPUP	Removes a popup menu from the menu bar
MENU INSERT ITEM	Inserts a menu item on a popup menu
MENU INSERT POPUP	Inserts a popup menu on the menu bar
(MENU) ITEM	Adds a menu item to a popup menu
(MENU) POPUP	Adds a new popup menu to the menu bar
(MENU) SEPARATOR	Adds a menu item separator to a popup menu
MENU UPDATE	Updates a previously defined menu option, or group of options
TOOLBARHIDE	Hides a specified toolbar
TOOLBARSHOW	Displays a specified toolbar

### Network Commands

SET CONNECTION*	Specifies a communications connector, including from third parties
SET NETID*	Changes the characters of the NetID variable

---

## Commands by Category, *continued*

---

### Print Commands

EDIT COPYSPECIAL*	Implements the <b>Copy Special</b> selection on the <b>Edit</b> menu
PRINT CANCEL	Terminates active print jobs
PRINT CLOSE	Closes the open print channel and terminates any active print jobs
PRINT FILE	Prints the specified file
PRINT FONT	Changes the active print font
PRINT NEWLINE	Sends a carriage return and line feed to the printer
PRINT NEWPAGE	Sends a form feed to the printer
PRINT OPEN	Opens a print channel
PRINT STRING	Prints the specified string
PRINT STYLE	Changes the active print character characteristics
PRINT TABS	Specifies the print tab width
PRINT TERMINAL*	Sends the incoming session window data to the printer
SELECTION PRINT	Prints the current selection
SET KEEPPRINTCHANNELOPEN	Relinquishes Windows print channel
SET PASSTHROUGH*	Sets the Raw Passthrough Mode for a printer

### Script Control Commands

CANCEL	Cancels script execution
COMPILE	Compiles the specified DCS script
DEBUG	Writes all script commands to the specified file as they execute
EXECUTE	Starts a script and does not return to the calling script
LINENUMBERS	Inserts linenumbers into a compiled script
NOSHOW	Suspends script command trace display
PERFORM	Starts a script and returns to the calling script when a RETURN executes
QUIT	Cancels all script execution and terminates the DCS session
RESTART	Branches execution to the first line of the executing script
SET RESULT	Assigns a value to the RESULT system variable
SHOW	Resumes command audit initiated by DEBUG command
SPAWN	Starts an independent script without interrupting current script
TASKERROR	Calls DCS's internal execution error handling routine

---

## Commands by Category, *continued*

---

### Session Window Commands

CLEAR	Clears the screen and the history buffer
DISPLAY	Displays a string in the session window (does not send the string to a remote system)
DISPLAYCONFIG	Changes the visual characters of a session window
EDIT COPYSPECIAL*	Implements the <b>Copy Special</b> selection on the <b>Edit</b> menu
KEYMAP LOAD	Loads the specified keymap
KEYMAP SAVE	Saves the specified keymap
NOSHOW	Suspends command audit initiated by <b>DEBUG</b> command
SCREEN	Specifies the size and position of the session window (shows and hides terminal; turns on and off updating)
SCROLL DOWN	Scroll down a specified number of lines in session history buffer
SCROLL LEFT	Scroll left a specified number of lines in session history buffer
SCROLL RIGHT	Scroll right a specified number of lines in session history buffer
SCROLL UP	Scroll up a specified number of lines in session history buffer
SET AUTOSCROLLTOCURSOR	Determines whether emulation scrolling follows the cursor
SET AUTOSIZE*	Determines the status of the <b>Automatically Size</b> check box on the <b>Display</b> tab of the <b>Session Properties</b> dialog
SET BACKSPACEDESTRUCTIVE*	Determines whether the backspace key is destructive
SET BUFFERLINES	Specifies the number of lines in the session window and its history buffer
SET COLUMNS	Specifies the number of columns in the session window
SET CURSOR	Sets the shape of the cursor and turns display on and off
SET DEFAULTSESSIONHANDLE	Specifies a handle for the default session window
SET TERMCLOSE	Allows or disallows you to close the session window
SET TERMFONT	Specifies the font displayed in the session window
SET WORDWRAP*	Specifies the column at which to wrap incoming text
WHEN SCREEN	Activates when the specified screen region is modified

### String Commands

COLLECT*	Stores incoming characters in a specified string
CONCAT	Concatenates two or more strings
DISPLAY	Displays a string in the session window (does not send the string to the remote system)

---

## Commands by Category, *continued*

---

PARSE	Locates a substring in another string, and stores preceding and succeeding characters from the string
PRINT STRING	Prints the specified string
SET DECIMAL	Specifies the number of decimal places for string representations of real numbers
WAIT STRING*	Pauses execution until the specified string is received
WHEN STRING*	Activates when the specified string is received

### System Commands

BEEP*	Sounds the default system .wav file a number of times
FKEYS	Shows and hides the function keys
KEY	Remaps a given key combination to the specified definition
KEYBOARD	Locks and unlocks the keyboard
KEYMAP RESET	Determines whether or not keystrokes are transmitted to the local or remote system
LEVEL	Specifies a given function key level
SET CONNECTMESSAGE*	Changes the connection message from the Phone Book
SET CONNECTRESULT*	Changes the value of the ConnectResult variable
SET FKEYSSHOW	Sets the function keys to appear when file opened
SET NETID*	Changes the characters of the NetID system variable
SET PASSWORD*	Changes the characters of the Password system variable
SET USERID*	Changes the characters of the UserID system variable
SYSTEM*	Performs a specialized system operation
TIMER RESET	Resets the specified timer
TITLE	Assigns a title, displayed in the application title bar, to a task file

### Table Commands

(DDE) TABLE REPLY	Sends a table to a DDE client in response to an ADVISE or REQUEST
(DDE) TABLE REQUEST	Requests data in the specified format from the DDE server
(DDE) TABLE SEND	Sends the specified table to the DDE server
RECORD FORMAT	Defines a virtual record template for session window data fields (used with the RECORD SCAN command)
RECORD READ	Reads data from a structured or text table into its record buffer
RECORD SCAN	Retrieves data from the session window into a table defined by the RECORD FORMAT command

---

## Commands by Category, *continued*

---

RECORD WRITE	Writes the contents of the record buffer to a structured or text table
TABLE CLEAR	Clears any existing data from the specified table
TABLE CLOSE	Closes the specified table
TABLE COPY	Copies the contents of a given table to the specified table
TABLE DEFINE	Prepares a table for random access file operation
TABLE LOAD	Imports data from a file or the clipboard to a table
TABLE SAVE	Exports the contents of the specified table to a file or the clipboard
TABLE SORT	Performs a sort on the specified table using the given criteria

### Telecommunications Commands

BREAK*	Sends a break signal to the remote system
CONNCONFIG	Sets a connector parameter for the active session
CONNECT	Establishes a connection with a remote system
DIAL*	Dials the specified number using the modem
DISCONNECT	Terminates a connection with a remote system
DROPDTR*	Holds the DTR line low for the current serial port
EMULCONFIG	Sets an emulator parameter for the active session
HANGUP*	Disconnects the telephone line on the modem
KERMIT COPY*	Copies a file on a remote host to another place on the remote host
KERMIT DIRECTORY*	Lists the contents of a directory on a remote system
KERMIT ERASE*	Deletes a file from directory on a remote system
KERMIT FINISH*	Sends a Kermit finish packet to the remote Kermit server
KERMIT FREESPACE*	Displays the number of free bytes on the remote system
KERMIT HELP*	Displays a summary of a topic on the remote system's operation
KERMIT LOGOUT*	Sends a Kermit logout packet to the remote Kermit server
KERMIT MESSAGE*	Sends a short message to an account on the remote system
KERMIT NEWDIRECTORY*	Changes the current working directory on a remote system
KERMIT RENAME*	Changes the name of a file on a remote system
KERMIT TYPE*	Displays the contents of a file from the remote system
KERMIT WHO*	Issues the KERMIT WHO command to the remote system
SENDERBREAK*	Sends a break signal to the remote system
SET BACKSPACEDESTRUCTIVE*	Determines whether the backspace key is destructive

---

## Commands by Category, *continued*

---

SET BACKSPACEKEY*	Specifies the character sent when the [Backspace] key is pressed
SET BAUDRATE*	Changes the baud rate of the selected communications port
SET BINARYTRANSFERPARAMS*	Changes the settings for a binary transfer protocol
SET BINARYTRANSFERS	Sets the current binary transfer protocol or host environment
SET CARRIERDETECT*	Sets the carrier detect flag
SET CONNECTION*	Specifies a communications connector, including from third parties
SET CONNECTMESSAGE*	Changes the connection message from the Phone Book
SET CONNECTRESULT*	Changes the value of the ConnectResult variable
SET DATABITS*	Sets the number of data bits to be used during transmission
SET EMULATION*	Loads the specified terminal emulation DLL
SET FLOWCONTROL*	Specifies the desired flow control method
SET LOCALECHO*	Shows and hides local keystrokes
SET NETID*	Changes the characters of the NetID system variable
SET OUTGOINGCR*	Appends line feed characters to outgoing carriage returns
SET PARITY*	Specifies the type of parity to be used during data transmission
SET PHONENUMBER*	Specifies the default phone number for the DIAL command
SET RETRY*	Specifies whether to continue redialing until connection
SET RETRYDELAY*	Sets the delay time between dial attempts
SET SIGNAL*	Allows the system bell to sound when a connection is made
SET SOUND	Controls whether warning bells from the host are enabled
SET STOPBITS*	Sets the number of stop bits used during serial transmissions
SET XCLOCK	Specifies how long to wait for an XCLOCK host message to clear before generating an execution error during a send
SET XSYSTEM	Specifies how long to wait for an XSYSTEM host message to clear before generating an execution error during a send
SETTINGS	Allows you to modify options on a <b>Session Properties</b> tab
WAIT ECHO	Pauses execution until DCS receives a character
WAIT PROMPT*	Pauses execution until a number of characters are received
WAIT QUIET	Pauses execution until an amount of time elapses without character transmission
WAIT SCREEN	Pauses execution until remote host changes the data in the terminal window
WAIT STRING*	Pauses execution until the specified string is received

---

## Commands by Category, *continued*

---

WHEN DISCONNECT	Activates when the communications connection is terminated
WHEN ECHO	Activates when any character is received
WHEN INPUT	Activates when a keystroke-generated character is transmitted
WHEN QUIET	Activates after time elapses without character transmission
WHEN STRING*	Activates when the specified string is received
XFERCONFIG	Sets a file transfer parameter for a session

### **Window Commands**

DISPLAYCONFIG	Changes the visual characters of a session window
SET APPTITLE	Displays the specified string in the application's title bar
SET WINDOWTITLE	Displays the specified string in the active window's title bar
WHEN WINDOW*	Filters out Windows messages to child windows
WINDOW ACTIVATE	Brings the specified window into focus
WINDOW ARRANGE	Tiles all open document windows
WINDOW CLOSE	Closes the specified window
WINDOW DEFAULT	Passes a Windows message intercepted by the WHEN WINDOW command to the default handler for a window
WINDOW HIDE	Hides the application window and its child window
WINDOW MAXIMIZE	Maximizes the specified window
WINDOW MESSAGE	Sends a Windows message to the specified window
WINDOW MINIMIZE	Sets DCS to run as an icon
WINDOW MOVE	Moves and resizes the specified window
WINDOW OPEN	Opens a new window, or one containing an existing file
WINDOW RESTORE	Restores the DCS application window to its prior state
WINDOW STACK	Stacks all open windows in cascade fashion
WINDOW UNHIDE	Displays the application window and its child window

---

# APPCONFIG

---

## APPCONFIG String

The APPCONFIG command is used to set the value of a parameter used to configure options which apply to the entire DCS application.

### Arguments

#### String

The String argument represents a single keyword followed by the assignment operator (=) and a valid setting. Together, the keyword and the setting are used to configure the options for the DCS application.

Keywords for General tab	Valid Setting(s)
ShowSplashScr	1 (true), 0 (false)
ConfirmMessages	1 (true), 0 (false)
SaveAppPosition	1 (true), 0 (false)
ShowErrorMessages	1 (true), 0 (false)
Use SessionWizard	1 (true), 0 (false)
ShowBkgrdBitMap	1 (true), 0 (false)
EnableOLEMenu	1 (true), 0 (false)
EnableOLEToolbar	1 (true), 0 (false)
EnableOLEStatusBar	true,false

Keywords for File Locations tab	Valid Setting(s)
DefaultSesEmulation	addsvp60, ansi, att605, att4425, ibm3270,ibm5250, tandem, vt420
DefaultSesConnector	comdir, comtapi, fmidll, merdnlnt, telnet
DefaultSesFileTransfer	indfile, kermit, wixfdll, zmodem
UserSessionDir	string of valid path
UserScriptDir	string of valid path
UserDownLoadDir	string of valid path
UserUpLoadDir	string of valid path
UserMemoDir	string of valid path
UserMapDir	string of valid path
UserConnectorDir	string of valid path
UserEmulationDir	string of valid path
UserFileTransferDir	string of valid path
UserDisplayDir	string of valid path
UserFontDir	string of valid path



---

## APPCONFIG, *continued*

---

### Comments

The keywords correspond to parameters available in the **General** and **File Location** tabs of the **Options** dialog . This command, therefore, allows you to set these parameters through the DCS script language rather than using the dialog .

The **ERROR** function returns **TRUE** if the **String** keyword is invalid.



**Note:** Changes made via scripting to application options remain in effect even after closing and restarting DCS.



Also see: **GETAPPCONFIG** function

### Example

In this example:

```
APPCONFIG "UseSessionWizard=false"  
APPCONFIG "UserSessionDir=p:\public\userssess"  
APPCONFIG "UserDownloadDir=c:\windows\profiles\user1\download"
```

the session wizard is disabled, the default session file directory is set to a mapped network drive location, and the default file download directory is set to a specific user's folder on a local drive.

---

# ARGUMENTS

---

## ARGUMENTS (Parameter, ...)

The ARGUMENTS command specifies the names of local variables a script routine uses.

### Arguments

(Parameter, ...)

The **(Parameter, ...)** argument specifies the local variables in a routine. It consists of zero or more string, numeric, or Boolean variables.

### Comments

If the ARGUMENTS command is the first command in the child routine, these variables may be used as parameters to pass values to and from a parent routine. When execution returns to the parent routine, all variables created by the ARGUMENTS command are destroyed.

You may define more than one ARGUMENTS command in a single routine. All ARGUMENTS commands may be used to create variables, but only the first one specified may pass parameters to and from the parent routine.

### Example

```
*subroutine
ARGUMENTS ($name, %amount, !value)
```

In this example, DCS creates three local variables that may also be used to pass parameters.

```
*subroutine
ARGUMENTS ()
ARGUMENTS ($str1, $str2, %index)
```

In this example, DCS creates three local variables. The first ARGUMENTS command carries a null parameter list (indicating the variables are not to be used as parameters, and will neither receive parameters from, nor pass parameters to, a parent routine).

---

# BEEP

---

## BEEP Num

The BEEP command rings the default system bell one or more times.



**Note:** This command does not apply to IBM TN3270 emulations.

### Argument

#### Num

The optional **Num** argument is an integer specifying the number of times DCS will ring the system bell. If the **Num** argument is not included, the system bell will ring once.

### Comments

If a sound driver is loaded, this command plays the default sound (or WAV file) through the PC's multimedia speaker. If your computer does not have multimedia capability, or if you have disabled the multimedia capability, the default PC system bell will ring from the PC's speaker.

### Example

In this example:

```
If Exists ("C:\TEXT.TXT")
  Display "OK"
Else
  Beep 3
```

if `TEXT.TXT` resides at the root of the `C :` drive, DCS displays **OK** in the active session window. If the file is not found, the bell rings three times.

---

## BEGIN

---

### BEGIN

The BEGIN command specifies the beginning of a command block. An END command must follow the command block.

#### Arguments

The BEGIN command takes no arguments.

#### Example

```
WHEN DISCONNECT
BEGIN
DISPLAY (0,0) "Connection terminated"
PERFORM no_carrier
END
```

The command block executes after the **WHEN DISCONNECT** command. DCS displays the string "Connection terminated" in the session window and then performs the routine labeled `no_carrier`. The **END** command specifies the end of the command block.

---

# BREAK

---

## BREAK DelayUnits

The BREAK command sends the break signal.



Note: This command does not apply to IBM TN3270 emulations.

### Arguments

#### DelayUnits

The optional **DelayUnits** argument is an integer specifying the length of the breaksignal, where one delay unit equals 117 milliseconds. If you do not include the **DelayUnits** argument, DCS uses a default value of two (specifying a short break).

### Comments

The representations for the standard break signals are:

Delay Units	Signal
2	Short Break (0.233 seconds)
30	Long Break (3.5 seconds)

### Example

In this example:

```
SEND "password"  
BREAK 30
```

DCS sends the string password to the remote system, followed by a long break.

---

# CANCEL

---

## CANCEL

The CANCEL command stops script execution.

### Arguments

The CANCEL command takes no arguments.

### Comments

The CANCEL command stops only one script; it will not stop any other script, even those launched by the stopped script.

### Example

In this example:

```
;create structured table0
TABLE DEFINE 0 FIELDS CHAR 20 INT 5 INT 10
;
;the following error routine checks to see if
;DCS was able to create table0, and if not,
;displays an error message in the current
;session window and terminates the script
IF ERROR
BEGIN
DISPLAY "Table not defined"
CANCEL
END
;
;fill structured table0 with data
TABLE LOAD 0 FROM "DATA" AS SYLK
;
;the following error routine checks to see if
;DCS was able to fill table0, and if not,
;displays an error message in the current
;session window and terminates the script
IF ERROR ()
BEGIN
DISPLAY "Table not loaded"
CANCEL
END
```

if either error occurs during script execution, the CANCEL command directs DCS to cancel script execution at that point. Using the CANCEL command in this manner ensures that script execution does not continue if critical script commands do not execute as intended.

---

# CLEAR

---

## CLEAR Screen WINDOW WinHandle

The CLEAR command clears the session window screen.

### Arguments

#### Screen

The optional **Screen** argument directs DCS to clear only the portion of the history buffer visible in the session window. If the **Screen** argument is not included, the entire history buffer of the session is cleared. When clearing a session window with paged terminal information from a remote system, the **Screen** argument is unnecessary since such sessions have no history buffer.

#### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular window in DCS.

The **WINDOW** clause clears the screen or history buffer of the session window specified by the **WinHandle** argument. Including this clause allows a script to clear any DCS window contents.

### Comments

If the **WINDOW** clause is not included in the command, the script clears the screen of the current session.

If the window handle in the clause is the handle of a script or memo window, the entire contents of the file will be deleted!

### Example

In this example:

```
DIAL
.
.
.
IF CONNECT ( )
CLEAR
ELSE
CANCEL
.
.
.
```

the script dials the phone number saved in the properties of the current session window. If the remote system allows DCS to connect, the contents of the session window and corresponding history buffer are deleted. If the remote system does not allow the connection to occur, the script stops.


---

## COLLECT

---

**COLLECT UNTIL String1 EXCLUDE String2 String3 LIMIT Num SAVECR SBoolean NOTERMINAL WINDOW WinHandle**

The COLLECT command stores a maximum of 254 incoming characters from a session window into a string variable until DCS receives a carriage return or line feed.

 **Note:** This command does not apply to IBM TN3270 emulations.

### Arguments

#### UNTIL String1

The optional **UNTIL** clause directs DCS to stop collecting characters if it receives the text contained in the **String1** argument prior to receiving a carriage return character (^M or hexadecimal 0x0D) or a line feed character (^J or hexadecimal 0x0A). The **String1** argument may contain wildcard characters.

#### EXCLUDE String2

The optional **EXCLUDE** clause directs DCS to exclude any characters contained in the **String2** argument from the characters it collects. DCS continues to collect and exclude characters until it encounters a line feed, carriage return, or the text in the **String1** argument.

#### String3

The **String3** argument is a string variable in which DCS stores the characters it collects.

#### LIMIT Num


The optional **LIMIT** clause directs DCS to stop collecting characters after it has received a number of characters equal to the integer in the **Num** argument. The **LIMIT** and **UNTIL** clauses can work together (DCS stops collecting characters when either condition is met).

#### SAVECR SBoolean

The optional **SAVECR** clause directs DCS to store carriage returns in the collection string based on the Boolean value of the **SBoolean** argument. If **SBoolean** is a true value, DCS will continue collecting text when it receives a carriage return. If **SBoolean** is a false value, DCS will stop collecting text when it receives a carriage return.

#### NOTERMINAL

The optional **NOTERMINAL** keyword directs DCS not to pass the collected characters to the terminal emulation for processing. As the characters are not passed to the terminal emulation, they are also not displayed in the session window.

 **Note:** The **NOTERMINAL** keyword should be used only in cases where the input to the COLLECT command is known. Otherwise, important terminal commands might be lost.

If you do not use the **NOTERMINAL** keyword, DCS passes the collected characters to the terminal emulation for processing by default. The **NOTERMINAL** keyword can extend the functionality of a terminal emulation to include processing for a customized remote command.



---

## COLLECT, *continued*

---

### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular window in DCS.

The **WINDOW** clause directs DCS to collect characters in the session specified by the **WinHandle** argument. The inclusion of this clause allows a script in DCS to collect characters from the specified session.

### Comments

If you do not include the **WINDOW** clause in the command, DCS collects characters in the active session window.

By default, DCS excludes the line feed character. If you want to include line feed characters in the collected text, you must employ the **EXCLUDE** clause, and the **String2** argument must contain a null string (“”).

DCS treats the **COLLECT** command implicitly like a **WAIT** command. DCS will not execute any commands or functions following the **COLLECT** command until the conditions of the **COLLECT** command are met.

### Example

In this example:

```
COLLECT $0
```

if the incoming string is Arnold Wilson (713) 555-1234, the variable \$0 assumes the value Arnold Wilson (713) 555-1234.

In this example:

```
COLLECT UNTIL "." $data
```

if the incoming string is Arnold Wilson .713.555.1234, the variable \$Data contains the characters “Arnold Wilson “. This string contains a space immediately after Arnold and immediately after Wilson.

---

## COLLECT, *continued*

---

In this example:

```
TABLE DEFINE 0 TEXT "MYFILE"
SET %donetime SECONDS (DATE (), 17:00:00)
WHILE SECONDS (DATE (), TIME ()) < %donetime
BEGIN
COLLECT @R0
RECORD WRITE 0
END
TABLE CLOSE 0
```

DCS collects each line of data coming into the session window and writes it to table zero until 5:00 P.M. At that time, the table is closed and saved to the text file MYFILE.

In this procedure:

```
*GETCHARS
WHEN STRING ``^[&o' RESUME
WAIT RESUME
;process $recv_chars
COLLECT $recv_chars NOTERMINAL
.
.
.
RETURN
```

all characters are passed to the terminal emulation for normal processing until the 'ESCAPE & o' (hexadecimal 1B 26 6F) sequence has been received. When this sequence is received, the WHEN STRING command is triggered, and execution passes to the COLLECT statement. DCS then collects all characters into the string variable \$recv\_chars until a CR (hexadecimal 0D) is received, but does not pass these characters to the terminal emulation.

---

# COMPILE

---

## COMPILE Script Make Display

The **COMPILE** command compiles a DCS script.

### Arguments

#### Script

The **Script** argument is a string specifying the name of the script to compile. You do not need to include the DCP file extension.

#### Make

The optional **Make** argument is specified by a Boolean operand. If **Make** is TRUE, the specified script is compiled only if the source script has been modified since the last compilation date. If **Make** is FALSE, or is not included, the specified script is compiled regardless of whether it has been modified since the last compilation.

#### Display

The optional **Display** argument is specified by a Boolean operand. If **Display** is TRUE, the normal compile dialog will appear during compilation. If **Display** is FALSE, or is not included, the script will be compiled without displaying the compile dialog box. If you use the **Display** argument, you must also use the **Make** argument.

### Comments

DCS will not execute any commands or functions following the **COMPILE** command until DCS completely compiles the script text file.

See also the **DEBUG** and **LINENUMBERS** commands.

### Example

This command:

```
COMPILE "tutorial"
```

directs DCS to compile the tutorial script. A successful compilation creates a task file named `TUTORIAL.DCT`.

This command:

```
COMPILE "tutorial" TRUE
```

directs DCS to compile the file `TUTORIAL.DCP` only if it has been modified since the last time it was compiled. A successful compilation creates a task file named `TUTORIAL.DCT`.

---

## CONCAT

---

### CONCAT StringVar String String ...

The CONCAT command concatenates (joins) an existing string with additional strings, placing the result in the existing string variable.

#### Arguments

##### StringVar

The StringVar argument is a string variable containing the characters to which other characters will be attached. This variable must exist before it is used as an argument in this command; it will not be created by the CONCAT command. After the CONCAT command executes, the variable specified as the StringVar argument will contain both its original contents and the content of any attached strings.

##### String

The String argument may be either another string variable or a literal string (text within quotes). At least one string must be included as an argument in this command in addition to StringVar .

##### String ...

Succeeding strings may be either string variables or string literals.

#### Example

```
$A = "Hello, "  
$name = "Larry, "  
CONCAT $A $name "How are you?"
```

This example concatenates the string variable \$A, the string variable \$name, and a literal string. DCS assigns the resulting string, `Hello, Larry, How are you?`, to the string variable \$A.

---

# CONNCONFIG

---

## CONNCONFIG String WINDOW WinHandle

The CONNCONFIG command is used to set the value of a parameter used in the connector configuration for the active session.

### Arguments

#### String

The String argument represents a single “keyword” followed by the assignment operator (=) and a valid setting. Together, the keyword and the setting are used to configure the connector for the active session.



**Note:** Configuration keywords for connectors shipped with client options, such as SNA Server, are available only if the client option has been installed.

Keywords for Modem	Valid Setting(s)
AREACODE	string
PHONENUMBER	string
COUNTRYCODE	string
MODEMTYPE	string

Keywords for Direct Serial	Valid Setting(s)
STOPBITS	1, 1.5, 2
PARITY	odd, even, mark, space, none
FLOWCONTROL	software, hardware, none
DATABITS	4, 5, 6, 7, 8
BAUDRATE	1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200
COMMPORT	COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9
PARITYCHECK	1 (true), 0 (false)
DROPDTR	1 (true), 0 (false)

Keywords for Meridian LAT32	Valid Setting(s)
SERVICENAME	string
NODE	string
PASSWORD	string
PORT	string
BLOCKIO	1 (true), 0 (false)

---

## CONNCONFIG, *continued*

---

Keywords for Telnet	Valid Setting(s)
HOSTNAME	string of host name, including node and socket
PORT	23
LINEMODE	yes, no
BINARYMODE	yes, no
TERMCUSTOM	none, selected, custom
AUTORECONNECT	yes, no
TERMTYPE	string of terminal type
WAITDATAMARK	yes, no
CRPAIRING	NULL, CRLF, NONE
SOCKETPORT	integer value of socket port
SERVICENAME	string of service name
BREAKTYPE	interrupt, telnet
NETWORKTYPE	TCP, SPX
BYPASS_TIMEWAIT	1 (true), 0 (false)

---

Keywords for SSH	Valid Setting(s)
SSHLEVEL	string indicating security level (2and1, 2only, 1only)

### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular child window.

The **WINDOW** clause directs DCS to apply the connector configuration settings to a particular session window.

### Comments

The keywords correspond to parameters available in the **Connectors** tab of the **Session Properties** dialog. This command, therefore, allows you to set these parameters through the DCS script language rather than using the dialog box.

The **ERROR** function returns **TRUE** if the **WinHandle** or **String** keyword is invalid.



**Note:** Although the Trace Play connector is shown on the **Connectors** tab of the **Session Properties** dialog, it cannot be configured via script.

If the Window Handle is not specified, DCS applies the connector configuration settings to the active session window.

---

## CONNCONFIG, *continued*

---

### Example

In this example:

```
%WINH = ACTIVE()  
SET DEFAULTSESSIONHANDLE %WINH  
SET CONNECTION "WINSOCK"  
CONNCONFIG "BINARY=YES"  
CONNCONFIG "DISPLAY ERRORS=NO"  
CONNCONFIG "PORT=19"
```

the connector of the active session window is set to Winsock, and several Winsock options are configured: binary negotiation is enabled, display error messages is disabled, and the connection port is set to 19.

### Example

In this example:

```
SET CONNECTION "OPENSSSH"  
CONNCONFIG "HOSTNAME=BUBBA"  
CONNCONFIG "SSHLEVEL=2ONLY"  
CONNECT
```

the connector of the active session window is set to SSH with the SSH level set to use level "2" security only.

---

# CONNECT

---

## CONNECT SessionFile WINDOW WinHandleVar

The CONNECT command establishes a connection with a host.

### Arguments

#### SessionFile

The optional **SessionFile** argument is a string specifying the path name of a session file.

#### WINDOW WinHandleVar

The optional **WinHandleVar** argument is an integer variable to which the CONNECT command assigns the window handle value of the active session window. In the multi-session environment of DCS, the window handles of sessions can be used in a DCS script to manipulate any of the sessions.



**Note:** The WinHandleVar variable must not have been previously assigned a value before the variable is used in the CONNECT command.

### Comments

The ERROR function returns TRUE if DCS cannot establish the connection.

When DCS performs the CONNECT command on a connected session, that session's window will become the active window.

The CONNECT command, when used alone, will both open and connect to the session file specified in the SessionFile argument. However, if you wish to open a session file with the LOAD command and then use the CONNECT command to connect the session later in the script, do not use the SessionFile argument with the CONNECT command. Instead, you must use the SET DEFAULTSESSIONHANDLE command to specify the target session.



**Note:** To enable session buttons and the status bar (top or bottom), you must explicitly enable them by using the DISPLAYCONFIG command (with appropriate keywords) prior to issuing the CONNECT command.

### Example

In this example:

```
LOAD "TSOSEND1.SES"  
%whnd=WINDOWHND("TSOSEND1")  
SET DEFAULTSESSIONHANDLE %whnd  
CONNECT
```

the session file TSOSEND1.SES is loaded before DCS attempts to connect to the host. In this case, the default sessionhandle must be specified. The CONNECT command will connect the session specified by the SET DEFAULTSESSIONHANDLE command.



---

## CONNECT, *continued*

---

This example:

```
$Session1="C:\DCSERIES\SETTINGS\MAIN.SES"  
;sets $Session1 to session file path  
CONNECT $Session1 Window %SessionHandle  
;starts session configured in MAIN.SES  
.  
.  
.  
;script uses other windows...  
CONNECT $Session1  
;makes window of $Session1 the active window
```

shows a method of activating a session window after a connection has been established and other operations performed.

---

# CONTINUE

---

## CONTINUE

The CONTINUE command branches execution to the WHILE command of a WHILE loop, and evaluates the Boolean argument.

### Arguments

The CONTINUE command takes no arguments.

### Comments

The CONTINUE command may only be executed from within the command block of a WHILE loop.

If the Boolean evaluates to FALSE, execution branches past the end of the WHILE loop without executing the command block. If the Boolean evaluates to TRUE, execution resumes at the first command of the command block.

### Example

In this example:

```
SET %ctr 0
WHILE %ctr < 10
BEGIN
RECORD READ 0
INCREMENT %ctr
IF %ctr % 2 = 0
CONTINUE
DISPLAY "Record " | STR %ctr | @R0
END
```

DCS reads ten records from Table 0 (zero) and displays only the odd numbered records. If the record number is even (the modulus of %ctr with respect to two equals zero), the CONTINUE command branches execution back to the WHILE command. The Boolean argument is again tested, and the next record is displayed if %ctr is less than ten.

---

# CREATE DIRECTORY

---

## CREATE DIRECTORY Path

The CREATE DIRECTORY command creates the specified directory.

### Arguments

#### Path

The Path argument is a string specifying the path of the directory to be created.

### Comments

If the directory already exists, or if the subdirectories leading to the directory do not exist, the CREATE DIRECTORY command fails and the ERROR function returns TRUE.

### Example

This script segment:

```
$TASKDIR = "C:\DCSERIES\TASK"  
CREATE DIRECTORY $TASKDIR  
SET DIRECTORY TASK $TASKDIR
```

creates the TASK directory in the DCSERIES directory on the C : drive and then sets the TASK directory as the default directory for the storage of script task files. However, if the DCSERIES directory does not exist or if the TASK directory already exists, the CREATE DIRECTORY command does not create a TASK directory.

---

## (DDE) ACCESS

---

### ACCESS Server Topic ChannelVar DDEList

The (DDE) ACCESS command initiates a DDE conversation between DCS and another Windows application (including another instance of DCS).

#### Arguments

##### Server

The **Server** argument is a string specifying the name of the server application.

##### Topic

The **Topic** argument is a string, in the server's terms, specifying the topic for the DDE conversation.

##### ChannelVar

The optional **ChannelVar** argument is an integer variable into which DCS stores a DDE channel identifier. The result assigned to this variable is interpreted according to the following table:

Result	Indication
0	No response
-1	An error occurred in establishing the conversation
-n	n multiple responses were received
n	A single conversation established, and n (an integer) is the channel identifier

If you do not include the **ChannelVar** argument, DCS can maintain only one DDE conversation at a time. If multiple DDE conversations are taking place, all DDE commands must refer to a proper channel number to ensure that they are communicating in the correct DDE conversation.

#### DDEList

The optional **DDEList** argument is a string variable into which each channel number is stored if DCS receives multiple responses (**ChannelVar** = -n). It is in the following format: "channel1, channel2, ..." If **ChannelVar** contains a value of 0, -1, or n, the **DDEList** argument will contain a null string.

#### Comments

The ERROR function returns TRUE if DCS could not establish a DDE conversation.

---

## (DDE) ACCESS, *continued*

---

### Example

In this example:

```
ACCESS "EXCEL" "EXPENSES.XLS" %channel
IF %channel <=0
BEGIN
DISPLAY (0,0) "Cannot establish conversation",
CANCEL
END
PERFORM dde_begin
```

DCS attempts to initiate a DDE conversation with Excel, using the `EXPENSES.XLS` work sheet as the topic. If the attempt fails (`%Channel = 0, -1, or -n`), the string "Cannot establish conversation" is displayed in the session window and execution cancels. If the attempt is successful, the routine labeled `dde_begin` is performed.

---

## (DDE) ACCESS CANCEL

---

### ACCESS CANCEL Channel

The (DDE) ACCESS CANCEL command terminates the active DDE conversation.

#### Arguments

##### Channel

The optional **Channel** argument is an integer specifying the channel value returned by the (DDE) ACCESS command. DCS does not require the **Channel** argument if only one DDE conversation is taking place.

#### Comments

If the **Channel** argument is not included during a session with multiple DDE conversations, DCS terminates all DDE conversations. However, DCS terminates only those conversations spawned by the script containing this command.

#### Example

In this example:

```
ACCESS "EXCEL" "EXPENSES.XLS"  
IF ERROR ()  
BEGIN  
  DISPLAY (0,0) "Cannot Access Excel"  
  CANCEL  
END  
PERFORM dde_request  
PERFORM dde_poke  
ACCESS CANCEL  
ACCESS "EXCEL" "SYSTEM"
```

a DDE conversion is initiated with Excel. If the initiation is successful, routines are performed that request data from, and poke data to, Excel. When these routines are complete, the DDE channel is terminated before another DDE channel is initiated.

---

## (DDE) INSTRUCT

---

### **INSTRUCT Channel Command, ...**

The (DDE) INSTRUCT command sends a string of commands to a DDE server. Once the commands have been sent, DCS waits for acknowledgment.

#### **Arguments**

##### **Channel**

The optional **Channel** argument is an integer specifying the channel value returned by the (DDE) ACCESS command. DCS does not require the **Channel** argument if only one DDE conversation is taking place.

##### **Command, ...**

The **Command** argument is a string specifying a list of instructions in the server's terms for the server to perform.

#### **Comments**

The ERROR function returns TRUE if the server could not perform the request.

#### **Example**

In this example:

```
ACCESS "EXCEL" "SYSTEM"  
INSTRUCT '[OPEN("C:\BUDGET.XLS") ]'
```

a DDE conversation is established with Excel that accesses the system. Excel then opens the BUDGET.XLS work sheet.

---

## (DDE) POKE

---

### POKE Data TO Channel Item

The (DDE) POKE command sends the specified data to the server in a DDE conversation and then waits for acknowledgment.

#### Arguments

##### Data

The **Data** argument is a string specifying the data to send, in the server's terms. If the string contains multiple fields of data, they should be tab delimited.

##### TO Channel Item

The **TO** clause specifies the destination of the data. The optional **Channel** argument is an integer specifying the channel value returned by the (DDE) **ACCESS** command. DCS does not require the **Channel** argument if only one DDE conversation is taking place.

The **Item** argument is a string specifying the desired data item.

#### Comments

You can use the (DDE) **POKE** command instead of the (DDE) **TABLE SEND** command when sending a single data item.

#### Example

In this example:

```
COLLECT $data
ACCESS "EXCEL" "EXPENSES.XLS" %channel
POKE $data TO %channel "R1C1"
ACCESS CANCEL %channel
```

a line of data coming into the session window is collected and sent to row 1, column 1, of the Excel spreadsheet `EXPENSES.XLS`.



---

## (DDE) REQUEST

---

### REQUEST DataVar FROM Channel Item

The (DDE) REQUEST command requests that data received from a server in a DDE conversation be placed in the specified variable, then waits for acknowledgment.

#### Arguments

##### DataVar

The DataVar argument is a string variable into which the requested data is stored.

##### FROM Channel Item

The FROM clause specifies the source of the requested data. The optional **Channel** argument is an integer specifying the channel value returned by the (DDE) ACCESS command. DCS does not require the **Channel** argument if only one DDE conversation is taking place.

##### Item

The Item argument is a string identifying a data item.

#### Comments

You may use the (DDE) REQUEST command instead of the (DDE) TABLE REQUEST command when only a single data item is needed.

During a DDE session between two instances of DCS, the (DDE) TABLE REQUEST command must be used to request data because DCS replies to requests via tables.

#### Example

In this example:

```
ACCESS "EXCEL" "BUDGET.XLS" %channel
REQUEST $totals FROM %channel "R1C1"
SET @R0 $totals
RECORD WRITE 0
```

the contents of row one, column one, of the Excel spreadsheet BUDGET.XLS are written to Table 0 (zero).

---

## (DDE) TABLE REPLY

---

### TABLE REPLY StructTableNum TO Channel Item

The (DDE) TABLE REPLY command sends the contents of a structured table in response to a DDE request (either an advise or a request) for data from the client, and then waits for acknowledgment.

#### Arguments

##### StructTableNum

The StructTableNum argument is an integer (from 0 to 15) specifying the number of a structured table containing the data to be sent.

##### TO Channel Item

The TO clause specifies the destination (in the client's terms) of the exported data. The optional **Channel** argument is an integer specifying the channel value returned by the (DDE) WHEN INITIATE command. DCS does not require the **Channel** argument if only one DDE conversation is taking place.

The **Item** argument is a string identifying the requested data item.

#### Comments

The ERROR function returns TRUE if the operation is unsuccessful.

#### Example

These commands:

```
WHEN POKE 0 TABLE 0 "item1"
BEGIN
TABLE REPLY 0 TO "POKE1"
END

WHILE TRUE
BEGIN
WAIT SIGNAL
END
```

service a (DDE) POKE from a DDE client on the topic POKE1.

---

## (DDE) TABLE REPLY, *continued*

---

In this example:

```
WHEN REQUEST 0 "item0"  
BEGIN  
  DISPLAY (0,0) "Request Activated"  
  TABLE LOAD 0 FROM "DATA1" AS SYLK  
  IF ERROR ()  
  BEGIN  
    DISPLAY "Can't load table"  
    CANCEL  
  END  
END  
TABLE REPLY 0 TO "item0"  
END
```

when DCS receives a data request from the DDE client on `item0`, DCS displays "Request Activated" in the session window, then loads the necessary data into structured Table 0 (zero) and sends the data to the client.

---

## (DDE) TABLE REQUEST

---

### TABLE REQUEST *StrucTableNum* FROM *Channel Item* AS *Format*

The (DDE) TABLE REQUEST command requests data from the DDE server, places the data into a structured table, and then waits for acknowledgment.

#### Arguments

##### *StrucTableNum*

The *StrucTableNum* argument is an integer (from 0 to 15) specifying the number of a structured table into which data is placed.

##### FROM *Channel Item*

The FROM clause specifies the source of the requested data (in the server's terms). The optional **Channel** argument is an integer specifying the channel value returned by the (DDE) WHEN INITIATE and (DDE) ACCESS commands. DCS does not require the **Channel** argument if only one DDE conversation is taking place.

The *Item* argument is a string identifying the requested data item.

##### AS *Format*

The AS clause specifies the format in which to store the imported data. The *Format* argument is specified by one of the following keywords:

DIF                      SYLK                      TEXT

#### Comments

The ERROR function returns TRUE if the server could not perform the request.

#### Example

In this example:

```
ACCESS "EXCEL" "BUDGET.XLS"
DISPLAY (0,0) "DATA REQUEST 0: "
TABLE REQUEST 0 FROM "item0" AS SYLK
IF ERROR ( )
  DISPLAY "Request denied"
ELSE
  DISPLAY (0,5) "Request Successful"
```

DCS initiates a DDE conversation with Excel and requests data from the BUDGET.XLS work sheet using *item0* as the topic.

---

## (DDE) TABLE SEND

---

### TABLE SEND *StrucTableNum* TO *Channel Item* AS *Format*

The (DDE) TABLE SEND command allows DCS to act as a DDE client. The command sends the contents of a structured table to the server in a DDE conversation and then waits for acknowledgment.

#### Arguments

##### *StrucTableNum*

The *StrucTableNum* argument is an integer (from 0 to 15) specifying the number of a structured table containing the data to be sent.

##### TO *Channel Item*

The TO clause specifies the destination, in the server's terms, of the exported data. The optional **Channel** argument is an integer specifying the channel value returned by the (DDE) ACCESS command. DCS does not require the **Channel** argument if only one DDE conversation is taking place.

The *Item* argument is a string identifying the requested data item.

##### AS *Format*

The AS clause specifies the standard data format in which to send the exported data. The *Format* argument is specified by one of the following keywords:

DIF                      SYLK                      TEXT

#### Comments

The ERROR function returns TRUE if the server could not perform the request.

---

## (DDE) WAIT SIGNAL

---

### WAIT SIGNAL

The (DDE) WAIT SIGNAL command pauses script execution until DCS receives the next DDE request from a client.

### Arguments

The (DDE) WAIT SIGNAL command takes no arguments.

### Comments

In order to process the entire current DDE request without interruption from the next incoming DDE request, DCS disables all incoming DDE requests while executing this command.

Any series of (DDE) WHEN commands intended to fulfill a DDE service request must end with this command.

### Example

These commands:

```
WHEN POKE 0 TABLE 0 "item0"  
PERFORM poke0  
  
WHEN POKE 1 TABLE 1 "item1"  
PERFORM poke1  
  
WHEN TERMINATE,  
BEGIN  
DISPLAY "(0,0) DDE link terminated"  
CANCEL  
END  
  
WHILE TRUE  
WAIT SIGNAL
```

prepare DCS to service an incoming (DDE) POKE from the DDE client. An infinite loop is established so that DCS waits to receive an incoming POKE each time execution returns from either the `poke0` or `poke1` routines. Execution terminates when DCS receives a terminate message from the client.

---

## (DDE) WHEN ADVISE

---

### WHEN ADVISE Index Channel Item Command

The (DDE) WHEN ADVISE command prepares DCS to service a DDE request to send continuous updates on a particular data item. DCS must be in a wait state to service a request.

#### Arguments

##### Index

The **Index** argument is an integer (from 0 to 16) specifying the command identifier. It allows multiple (DDE) WHEN ADVISE commands to be active at the same time. DCS may have a maximum of 16 (DDE) WHEN ADVISE commands active at the same time.

**Index** argument 16 is a unique index. When DCS receives an ADVISE message, it will look through indexes 0 through 15 first. If no corresponding WHEN ADVISE message is found, the message is placed in the **Item** string specified with the command. For example, the script line:

```
WHEN ADVISE 16 $item DISPLAY "ADVISE-Unknown item, \  
" | $item | "^M"
```

signals any unknown message, displaying only those which were not found in the first 16 indexes (0 through 15).

##### Channel

The optional **Channel** argument is an integer specifying the channel value returned by the (DDE) WHEN INITIATE command. DCS does not require the **Channel** argument if only one DDE conversation is taking place.

##### Item

The **Item** argument is a string identifying, in the client's terms, the requested data item.

##### Command

The **Command** argument specifies a logical command (either a single command or a command block). Activation of the (DDE) WHEN ADVISE command executes the logical command.

#### Comments

The (DDE) WHEN ADVISE command is activated when DCS receives either an advise message (ADVISE is set to TRUE) or an unadvise message (ADVISE is set to FALSE), and the data item requested matches the value of the **Item** argument.

DCS determines if the (DDE) WHEN ADVISE command was activated by an advise or an unadvise message with the ADVISE function. Script commands issued to fulfill an advise request should be disabled when an unadvise message is received.

Client requests for data should be fulfilled using the (DDE) TABLE REPLY command.

---

## (DDE) WHEN ADVISE, *continued*

---

### Example

In this example:

```
TABLE DEFINE 3 FIELDS CHAR 20
WHEN ADVISE 0 "item0"
PERFORM advise

WHILE TRUE
WAIT SIGNAL

*advise
IF NOT ADVISE
BEGIN
DISPLAY (0,0) "Advise canceled by client"
WHEN CANCEL SCREEN 0
RETURN
END

WHEN SCREEN 0 (24 10 1)
BEGIN
PERFORM grabItem0 (3)
TABLE REPLY 3 TO "item0"
END
RETURN
```

by executing an **ADVISE** command, the client requests DCS to send an update whenever the specified data item is modified. You must establish criteria to determine when an update should be sent.

In this example, the script looks for a particular region of the screen to be modified, indicating that there is updated information to send to the client with the (DDE) **TABLE REPLY** command. When DCS receives an unadvise message, the string "Advise canceled by client" is displayed in the session window and the **WHEN SCREEN** command is canceled.



---

## (DDE) WHEN EXECUTE

---

### WHEN EXECUTE Channel CommandVar Command

The (DDE) WHEN EXECUTE command prepares DCS to service an execute request from the DDE client. DCS must be in a wait state to service the request.

#### Arguments

##### Channel

The optional Channel argument is an integer variable specifying the channel value returned by the (DDE) WHEN INITIATE command. DCS does not require the Channel argument for only one DDE conversation.

##### CommandVar

The CommandVar argument is a string variable which stores the command specified by the client.

##### Command

The Command argument specifies a logical command (either a single command or a command block). Activation of the (DDE) WHEN EXECUTE command executes the logical command. The command established in this argument should branch execution according to the result returned in the CommandVar argument.

#### Comments

Since DCS's script language is compiled, the DDE client cannot directly add functionality to a compiled script. Commands sent from the server must be parsed through script. Routines should be established that will branch execution according to the command sent.

---

## (DDE) WHEN EXECUTE, *continued*

---

### Example

In this example:

```
WHEN INITIATE %ch PERFORM dde_initiate

WHEN EXECUTE %ch $command
PERFORM dde_parse_cmd

WHEN TERMINATE
BEGIN
DISPLAY `DDE ended`
RETURN
END

WHILE TRUE
WAIT SIGNAL
CANCEL

*dde_parse_cmd
DISPLAY $command
IF $command = "WINDOW STACK"
WINDOW STACK
ELSE
IF $command = "CLEAR SCREEN"
CLEAR SCREEN
ELSE
IF $command = "LAUNCH `PAINT.EXE`"
LAUNCH "PAINT.EXE"
ELSE
RETURN
```

a series of nested **IF** commands determine which action is performed, depending on the contents of the `$command` variable.

---

## (DDE) WHEN INITIATE

---

### WHEN INITIATE ChannelVar Command

The (DDE) WHEN INITIATE command is activated when DCS is in a wait state and a new DDE conversation is established.

#### Arguments

##### ChannelVar

The optional **ChannelVar** argument is a numeric variable into which DCS will store a DDE channel identifier generated by the client.

##### Command

The **Command** argument specifies a logical command (either a single command or a command block). Activation of the (DDE) WHEN INITIATE command executes the logical command.

#### Comments

DCS is in a wait state after DCS has executed a WAIT command and before the condition of the WAIT command is fulfilled.

#### Example

In this example:

```
WHEN INITIATE PERFORM dde_initiate
WHEN ADVISE PERFORM dde_advise
WHEN TERMINATE DISPLAY 'DDE ended', RETURN

WHILE TRUE
WAIT SIGNAL
```

DCS prepares to perform the routine `dde_initiate` when a DDE conversation is established. DCS then pauses execution at the (DDE) WAIT SIGNAL command, waiting for a DDE initiate message. Once received, execution returns to the (DDE) WAIT SIGNAL command, while DCS waits to receive an advise message.

Each time DCS receives an advise message, it performs the `dde_advise` routine and then returns to the (DDE) WAIT SIGNAL command until the DDE conversation is terminated.

---

## (DDE) WHEN POKE

---

### WHEN POKE Index TABLE Table Channel Item Command

The (DDE) WHEN POKE command prepares DCS to service a DDE request message from the client to receive a particular data item. DCS must be in a wait state to service the request. Received data is placed in the specified table.

#### Arguments

##### Index

The **Index** argument is an integer (from 0 to 16) specifying the command identifier. It allows multiple (DDE) WHEN POKE commands to be active and to access the same table at the same time. DCS may have a maximum of 16 (DDE) WHEN POKE commands active at one time.

**Index** argument 16 is a unique index. When DCS receives a POKE message, it will look through indexes 0 through 15 first. If no corresponding WHEN POKE message is found, the message is placed in the **Item** string specified with the command. For example, the script line:

```
WHEN POKE 16 $item DISPLAY "POKE-Unknown item, "\
| $item | "^M"
```

signals any unknown message, displaying only those which were not found in the first 16 indexes (0 through 15).

With **Index** 16, the **Item** string may be used as the switch variable for the SWITCH command. Since the SWITCH command compares the switch variable to an unlimited number of case statements, you can effectively handle more than 16 messages. See the DDE Server example.

##### TABLE Table

The **TABLE** clause specifies the table in which to place the data. The **Table** argument is an integer (from 0 to 15) specifying the table number.

##### Channel

The optional **Channel** argument is an integer specifying the channel value returned by the (DDE) WHEN INITIATE command. DCS does not require the **Channel** argument if only one DDE conversation is taking place.

##### Item

The **Item** argument is a string identifying, in the client's terms, the requested data item.

##### Command

The **Command** argument specifies a logical command (either a single command or a command block). Activation of the (DDE) WHEN POKE command executes the logical command.

---

## (DDE) WHEN POKE, *continued*

---

### **Comments**

When DCS receives a poke message, and the data item requested matches the value of the Item argument, the (DDE) WHEN POKE command is activated.

### **Example**

See the (DDE) WAIT SIGNAL command.

---

## (DDE) WHEN REQUEST

---

### WHEN REQUEST Index Channel Item Command

The (DDE) WHEN REQUEST command prepares DCS to service a DDE request for a particular data item. DCS must be in a wait state to service this request.

#### Arguments

##### Index

The **Index** argument is an integer (from 0 to 16) specifying the command identifier. It allows multiple (DDE) WHEN REQUEST commands to be active at the same time. DCS may have a maximum of 16 (DDE) WHEN REQUEST commands active at one time.

**Index** argument 16 is a unique index. When DCS receives an **REQUEST** message, it will look through indexes 0 through 15 first. If no corresponding **WHEN REQUEST** message is found, the message is placed in the **Item** string specified with the command. For example, the script line:

```
WHEN REQUEST 16 $item DISPLAY "REQUEST-Unknown item, \  
" | $item | "^M"
```

signals any unknown message, displaying only those which were not found in the first 16 indexes (0 through 15).

##### Channel

The optional **Channel** argument is an integer specifying the channel number returned by the (DDE) WHEN INITIATE command. DCS does not require the **Channel** argument if only one DDE conversation is taking place.

##### Item

The **Item** argument is a string identifying, in the client's terms, the requested data item.

##### Command

The **Command** argument specifies a logical command (either a single command or a command block). Activation of the (DDE) WHEN REQUEST command executes the logical command.

#### Comments

When DCS receives a request message, and the data item requested matches the value of the **Item** argument, the (DDE) WHEN REQUEST command is activated.

DCS is in a wait state after DCS has executed a **WAIT** command and before the condition of the **WAIT** command is fulfilled. Client requests for data should be filled using the (DDE) **TABLE REPLY** command.

---

## (DDE) WHEN REQUEST, *continued*

---

### Example

In this example:

```
TABLE DEFINE 0 FIELDS INT 10 INT 10
WHEN REQUEST 0 "item0"
BEGIN
DISPLAY (0,0) "Request activated"
TABLE REPLY 0 TO "item0"
IF ERROR ()
BEGIN
DISPLAY (1,0) "Reply failed"
END
END
WHILE TRUE
WAIT SIGNAL
```

DCS prepares to service a request from the client. When the request is received, a message is displayed in the session window. DCS fills this request using the (DDE) TABLE REPLY command and execution returns to the (DDE) WAIT SIGNAL command.

---

## (DDE) WHEN TERMINATE

---

### WHEN TERMINATE Channel Command

The (DDE) WHEN TERMINATE command is used when DCS is acting as a DDE server or a DDE client. It is activated when DCS is in a wait state and the current DDE conversation is terminated.

#### Arguments

##### Channel

The optional **Channel** argument is an integer specifying the channel number returned by the (DDE) WHEN INITIATE command (DCS acting as a server) or (DDE) ACCESS command (DCS acting as a client). DCS does not require the Channel argument if only one DDE conversation is taking place.

##### Command

The **Command** argument specifies a logical command (either a single command or a command block). Activation of the (DDE) WHEN TERMINATE command executes the logical command.

#### Example

See the (DDE) WAIT SIGNAL command.



---

## DEBUG

---

### DEBUG FileName

As each command or function executes, the **DEBUG** command writes the command or function to a file to aid you in solving problems with a script.

#### Arguments

##### FileName

The **FileName** argument is a string naming the file to which DCS writes the output. The file name may be up to twelve characters long (including the path, name, period and extension) and must specify a valid file name for your system. For best results, create the debug file in a root directory (e.g., `c:\debug.txt`).

If the file does not exist, DCS will create the file before it executes the **DEBUG** command and will place the file into the task directory of the executing script. If the file exists prior to the execution of the **DEBUG** command, DCS will delete the previous contents of the file before executing the command.

If you use a question mark (?) for the **FileName** argument, and if the **SHOW** command follows the **DEBUG** command, DCS will enter step mode. While in step mode, DCS displays the currently executing command in the Script Compiler dialog. This dialog also contains a Stop button and a Pause or Resume button.

#### Comments

DCS stops placing text in the debug file, when it encounters the **NOSHOW** command, and will resume writing text to the file when it encounters the **SHOW** command. You can strategically place the **SHOW** and **NOSHOW** commands in a script to allow you to debug portions of a script rather than all of it.

DCS executes only the first **DEBUG** command it encounters in a script. It ignores any subsequent **DEBUG** commands in the script, or in the subroutines the script may execute.

---

## DEBUG, *continued*

---

### Example

In this example:

```
DEBUG "DATAREAD.BUG"
SHOW

*begin_read
RECORD READ 0 at 0
WHILE NOT EOF ( )
BEGIN
DISPLAY @R0.1 | " - " | @R0.5
RECORD READ 0
END
PERFORM end_read

*end_read
DISPLAY "End of file reached"
NOSHOW
CANCEL
```

the **DEBUG** command will write its output to the file **BUGFILE**. When the **SHOW** command executes, DCS begins collecting the commands and functions as they execute, copying each script line to **BUGFILE**. Each time the commands between the label **\*begin\_read** and the line **DISPLAY @R0.1 | " - " | @R0.5** execute, DCS adds them to the debug file. When the end of the file is reached, script execution branches to the **\*end\_read** label. The script line **DISPLAY "End of file reached"** executes and is written to the debug file. The **NOSHOW** command executes, and any further commands executed are not written to **BUGFILE**.

---

# DECREMENT

---

## DECREMENT IntVar

The DECREMENT command decreases the value of the specified numeric variable by one.

### Arguments

#### IntVar

The IntVar argument specifies the integer variable to be affected.

### Comments

An integer variable must be specified (a real variable will generate a syntax error).

### Example

In this example:

```
SET %num 10
WHILE % num > 0
BEGIN
RECORD READ 0
DISPLAY (%num, 0) @R0
DECREMENT %num
END
```

the command DECREMENT %num is used as a counter so that exactly ten records are read and displayed.


---

## DIAL

---

### DIAL PhoneNum RETRY Retries DELAY RetryDelay

The DIAL command directs the modem to dial a phone number. It is equivalent to selecting **Connect** on the **Session** menu when using the Modem (TAPI) connector.

 **Note:** The DIAL command does not apply to IBM TN3270 emulations.

#### Arguments

##### PhoneNum

The optional **PhoneNum** argument is a string specifying the phone number to dial. If it is not included, the modem dials the phone number specified in the active session file. If no phone number is specified in the active session file, the modem dials the phone number specified in the last SET PHONENUMBER command. If no SET PHONENUMBER command has been executed DCS will return an error (since it does not have a phone number to dial).

##### RETRY Retries

The optional RETRY clause directs DCS to redial the specified phone number if the connection is initially unsuccessful. If it is not included, the phone number is redialed according to the active session file. The **Retries** argument is a numeric specifying the number of times to dial. Setting the value of **Retries** to -1 directs DCS to continuously redial until a successful connection is made.

##### DELAY RetryDelay

The optional **DELAY** clause determines the amount of time for DCS to wait to make a successful connection before retrying to dial. If it is not included, the retry delay is determined by the active session file, where the default value is 30 seconds. The **RetryDelay** argument is a numeric specifying the delay in seconds.

If you include the **DELAY** clause, you must also include the **RETRY** clause.

#### Comments

If different phone number settings are specified in the active session file, the DIAL command temporarily overrides these settings, but does not modify the session file.

Due to the way the Windows TAPI operates (with its parameters separated and specific rules for dialing), problems may arise when the area code and/or country code parameters are defined in the TAPI configuration, and then also defined in the DIAL command. To avoid these problems, set the area code, country code and phone number using the CONNCONFIG command and then use the DIAL command without any parameters. See the CONNCONFIG command for the valid keyword parameters for the Modem connector.

DCS treats the DIAL command like the Connect option in the Session menu. For example, if DCS has connected to another computer, executing the DIAL command will have no effect.

---

## DIAL, *continued*

---

### Examples

This example:

```
WINDOW OPEN SETTINGS "" %HWND
SET DEFAULTSESSIONHANDLE %HWND
SET CONNECTION "TAPI"
CONNCONFIG "AREACODE=281"
CONNCONFIG "PHONENUMBER=555-5555"

DIAL
```

directs DCS to set the areacode and phone number parameters in the session file and then dial.

In this example:

```
LOAD "COMPUSER.SES"
IF CONNECT ()
BEGIN
SEND "/off"
HANGUP
END

DIAL "555-1234" RETRY 3
WAIT STRING ">"
SEND NOCR "^M"
WAIT STRING "ID:"
SEND "secretid"
```

DCS loads a session file named `COMPUSER.SES`. DCS then checks to see if it is presently connected and, if so, logs off and hangs up. DCS then dials the specified phone number, waits for the expected prompts and sends the necessary responses.

---

## DIALOG

---

**DIALOG** (x, y, w, h) Title SYSMENU SBoolean MODAL MBoolean NOFOCUS  
FBoolean

DialogIndex

Option

Option...

### DIALOG END

The DIALOG command creates custom dialog boxes prompting your interaction with DCS. One or more dialog options may follow the DIALOG command. A DIALOG END command indicates the end of the dialog box definition and must follow the DIALOG command and the dialog options.



**Note:** Dialog boxes in the script language are governed in most cases by the operating system (Windows 95/98/NT). Therefore, settings which affect the display of dialog boxes in Windows will also affect those dialogs created with the script language. For example, if the display settings are set to Large Fonts, scripted dialog boxes will use large fonts and, as a result, be larger than dialog boxes displayed when Windows is using Small Fonts.

### Arguments

(x, y, w, h)

The optional coordinate set (x, y, w, h) specifies the desired position and size of the dialog box. It indicates the top left corner (x, y), width (w), and height (h). The coordinates are specified as logical units:

- ▼ Horizontally, there are four logical units per character
- ▼ Vertically, there are eight logical units per line.

The (x, y) coordinates for a dialog box are relative to the top left corner of the application window, which is (0,0).

If the coordinate set is included, one to four coordinates can be specified (for example, (, ,40,) specifies width only). If you do not include a coordinate set, DCS assigns a default size and placement.

Dialog box coordinates are intended to be machine-independent and are, therefore, defined in terms of system character and line widths, instead of pixels. This allows for the best appearance of a dialog on different types of machines and monitors.

### Title

The optional **Title** argument is a string which will appear in the title bar of the dialog box. DCS will also include a Control, or System, menu with the title bar. If you do not include the **Title** argument and do not include the SYSMENU clause (or if you have included the SYSMENU clause with a false Boolean value as its argument), DCS will not display a title bar (or a title) for the dialog box. When you define dialog boxes with the **Title** argument, you can move the dialog box around the screen by dragging the title bar. If the **Title** argument contains a null string (“”), the dialog box appears with a blank title bar.

---

## DIALOG, *continued*

---

The **SYSTEMMENU**, **MODAL**, and **NOFOCUS** clauses are order-dependent; however, you do not have to include all three clauses in the command. For example, if you include the **SYSTEMMENU** and **NOFOCUS** clauses, but not the **MODAL** clause, the **NOFOCUS** clause must follow the **SYSTEMMENU** clause.

### **SYSTEMMENU SBoolean**

The optional **SYSTEMMENU** clause controls the display of the dialog box's control menu. The **SYSTEMMENU** clause is incompatible with the **MODAL** clause. The optional **SBoolean** argument is a Boolean value.

If you do not include the **SBoolean** argument with the **SYSTEMMENU** clause, or if **SBoolean** evaluates to a true Boolean value, DCS will create a dialog box with a control menu. If you include the **SBoolean** argument and it evaluates to a false Boolean value, the dialog box will not have a control menu.

### **MODAL MBoolean**

The optional **MODAL** clause allows you to specify whether a dialog box is a non-modal or a modal dialog. If you do not include the **MODAL** clause, the **DIALOG** command will define a non-modal dialog. When a dialog box is non-modal, you are allowed to select options outside of the dialog box (like the menus or windows in DCS or other Windows applications). However, if you include the **MODAL** clause, you are not allowed to select any options outside of the dialog box (or possibly in any other application), until the script cancels the dialog box. Usually, you must interact with some option in the dialog box in order to cancel the dialog.

The **MODAL** clause and its optional **MBoolean** argument can make a dialog box either application modal or system modal. If the dialog box is an application-modal dialog box, you are allowed to interact with other applications, but you are not allowed to interact with other options in DCS, until the script cancels the dialog box. If the dialog box is a system-modal dialog, you are not allowed to interact with any other application or any other option in DCS, until the script cancels the dialog box.

The **MBoolean** argument is an optional argument to the **MODAL** clause and is a Boolean value. The **MODAL** clause will create an application modal dialog box when you do not include the **MBoolean** argument or when the **MBoolean** argument evaluates to a false Boolean value. If the portion of the script dealing with an application modal dialog box has an error in its logic, the Script menu (and other DCS menus) might become inaccessible.

The **MODAL** clause will create a system modal dialog box when you include the **MBoolean** argument and it evaluates to a true Boolean value. If the portion of the script dealing with a system modal dialog box has an error in its logic, the Program Manager (and other applications) might become inaccessible.

### **NOFOCUS FBoolean**

The optional **NOFOCUS** clause determines whether the dialog box will become the active window when initially displayed. When a dialog or window is active, it has a highlighted title bar and border; other open windows or dialogs will have dimmed title bars and borders. The optional **FBoolean** argument is a Boolean value. If you do not include the **FBoolean**

---

## DIALOG, *continued*

---

argument, or if it evaluates to a false Boolean value, the **NOFOCUS** clause will create a dialog and make it the active window. When you include the **FBoolean** argument, and the argument evaluates to a true Boolean value, the dialog box created will not be the active window. When you click the mouse button in the dialog box, DCS will make the dialog box the active window.

### DialogIndex

The optional **DialogIndex** argument is an integer (from 0 to 15) identifying a particular dialog box. If it is not included, DCS uses zero as the default value. If two dialog boxes in the same script have an identical **DialogIndex** value, DCS will destroy an existing dialog box in favor of a newly created dialog box.

### Option

Each **Option** argument is actually another script command which specifies a dialog control or command. Dialog controls are either active or static. You are allowed to select an active control in a dialog box, but not a static control. Static controls only display information; they are not interactive. The following are the valid script commands for dialog boxes:

BUTTON	ICON	LISTBOX	RADIOBUTTON
CHECKBOX	ICONBUTTON	MESSAGE	RADIOGROUP
EDITTEXT	GROUPBOX	PICTURE	WIDEBUTTON

### Comments

DCS treats active dialog box controls like **WHEN** commands.

The following limitations apply to dialog box controls:

Control	Maximum Quantity
Total Controls in a Dialog Box	255
Total Buttons, Wide Buttons, & Icon Buttons	128
Total Check Boxes	16
Total Edit Text Boxes	64
Total List Boxes	10
Total Pictures	1
Total Radio Groups	16

You may include a maximum of 255 controls per dialog box; however, the amount of available memory in your computer and design considerations (such as the practicality and ease of interacting with a dialog) might limit the number of controls you put into a dialog box to fewer than 255.

DCS can display more than one dialog box at a time. DCS will stop displaying a dialog box when a script encounters one of the following: the **DIALOG CANCEL** command, the end of the script, or the **DIALOG** command using a **DialogIndex** identical to that of a dialog box currently displayed.

When you create a dialog box with the **DIALOG** command, you can close the dialog in a num-



---

## DIALOG, *continued*

---

ber of ways. Consider the following script segment:

```
DIALOG
Button "Cancel" Resume
DIALOG END
WAIT RESUME
DIALOG CANCEL
```

This dialog box will disappear when you click the button titled `Cancel`. However, if you want to close a dialog box through its control menu or through the `[ALT]+[F4]` key combination, you must add a title bar to the dialog box and must add the `CANCEL` keyword to one of the buttons that you define for the dialog box. The following script segments are based on the previous script sample:

```
DIALOG "The Title of the Dialog Box"
BUTTON CANCEL "Cancel" RESUME
DIALOG END
Wait Resume
DIALOG CANCEL
```

or

```
DIALOG SYSMENU
Button CANCEL "Cancel" Resume
DIALOG END
WAIT RESUME
DIALOG CANCEL
```

In these examples, you can close the dialog box by selecting the `Cancel` button in the dialog box, by entering the `[ALT]+[F4]` key combination, by entering the `[ESC]` key (adding the `CANCEL` keyword allows you to use the `[ESC]` key), by selecting the `Close` option in the control menu of the title bar, or by double-clicking on the control menu.

The control menu will only appear in a dialog if you include the `Title` argument or the **SYSMENU** clause (the `SYSMENU` keyword only or the keyword with a true value for the **SBoolean** argument); however, if you want the control menu in the dialog box, but you do not want text in the title bar, compose the `Title` argument as a null string (`""`) or include the **SYSMENU** clause without the `Title` argument.

---

## DIALOG, *continued*

---

### Example

This script segment:

```
DIALOG "Hello" SYSMENU FALSE MODAL TRUE NOFOCUS \  
TRUE MESSAGE "Please respond!!"  
BUTTON DEFAULT "&OK" RESUME  
DIALOG END  
WAIT RESUME  
DIALOG CANCEL
```

displays a dialog box that requires a response before performing other actions on the PC.

For other examples, see each individual dialog box command following this section.

---

## (DIALOG) BUTTON

---

### **BUTTON** (x, y, w, h) **Default Title Command**

The (DIALOG) BUTTON dialog control command displays a button control in a dialog box.

#### **Arguments**

##### **(x, y, w, h)**

The optional coordinate set **(x, y, w, h)** specifies the top left corner (x, y), width (w), and height (h) of the button. The coordinates are in logical units. See the DIALOG command to determine the proper logical unit dimensions for your system. The (x, y) coordinates are relative to the dialog box which contains the button. The top left corner of the dialog box is (0,0).

If the coordinate set is included, one to four coordinates can be specified (for example, (, ,40,) specifies width only). If you do not include a coordinate set, DCS assigns a default size and placement.

#### **Default**

The optional **Default** argument is specified by one of the following two keywords:

<b>Keyword</b>	<b>Action</b>
DEFAULT	Allows the button to be automatically clicked when the [RETURN] key is pressed, unless another button is highlighted.
CANCEL	Causes the button to be automatically clicked when the [ESC] key is pressed, regardless of which button is highlighted.

A default button is displayed in the dialog box with a black border. Only one DEFAULT button may be specified per dialog box.

#### **Title**

The **Title** argument is a string specifying the characters to display on the button.

Preceding a character in the **Title** argument by an ampersand (&) causes the character to appear underlined on the button. The button can then be clicked by pressing the [ALT] key and the key of the underlined character simultaneously.

#### **Command**

The optional **Command** argument specifies a logical command (either a single command or a command block). Clicking the button executes the logical command. If a command is not included, clicking the button has no effect.

#### **Comments**

A button is an active control. You may include a maximum of 128 buttons (including buttons, icon buttons, and wide buttons) in a single dialog box.

---

## (DIALOG) BUTTON, *continued*

---

### Example

```
Dialog
CHECKBOX 0 "Initially Unchecked"
BUTTON (,,38,15) "1st Button" RESUME
BUTTON (,,38,15) DEFAULT "Default" RESUME
Dialog End

Wait Resume
DIALOG CANCEL
```

---

## DIALOG CANCEL

---

### DIALOG CANCEL DialogIndex

The DIALOG CANCEL command removes the active dialog box from the screen.

#### Arguments

##### DialogIndex

The optional DialogIndex argument is an integer (from 0 to 15) identifying a particular dialog box. If it is not included, DCS uses zero as the default value.

If you specify a value of -1, the DIALOG CANCEL command will remove all existing dialogs created by the script in which the command appears.

#### Comments

DCS removes a dialog box from the screen when script execution ends, or upon executing a DIALOG CANCEL command.

#### Example

In this example:

```
DIALOG
MESSAGE "Preparing for Text Capture..."
BUTTON DEFAULT "Continue" RESUME
BUTTON CANCEL "Abort" CANCEL
DIALOG END
WAIT RESUME
DIALOG CANCEL
PERFORM text_receive
```

the DIALOG CANCEL command removes the dialog box from the screen before the text transfer begins, enabling you to see the entire session window.

---

## (DIALOG) CHECKBOX

---

### CHECKBOX (x, y, w, h) Default CheckBoxText Command

The (DIALOG) CHECKBOX dialog control command displays a check box control in a dialog box.

#### Arguments

##### (x, y, w, h)

The optional coordinate set (x, y, w, h) specifies the top left corner (x, y), width (w), and height (h) of the check box (box and text). Specifying a width and height for a check box will not actually change the size of the check box, but will restrict or enlarge the space allocated to the display of the check box. See the DIALOG command to determine the proper logical unit dimensions for your system. The (x, y) coordinates are relative to the dialog box which contains the check box. The top left corner of the dialog box is (0,0).

If the coordinate set is included, one to four coordinates can be specified (for example, (, ,40,) specifies width only). If you do not include a coordinate set, DCS assigns a default size and placement.

#### Default

The optional **Default** argument is an integer specifying the initial state of the check box:

Integer	Initial State
0 (zero)	Indicates the initial state is unchecked
1 (one)	Indicates it is checked

If the default argument is not included, the default state is unchecked.

#### CheckBoxText

The CheckBoxText argument is a string describing the check box option. It is displayed to the right of the check box.

#### Command

The optional **Command** argument specifies a logical command (either a single command or a command block). Changing the state of the check box executes the logical command. If a command is not included, changing the state of the check box has no effect.

#### Comments

A check box is an active control. You can obtain the state of a check box using the (DIALOG) CHECKBOX function. You may include a maximum of 16 check boxes in a single dialog box.

---

## (DIALOG) CHECKBOX, *continued*

---

### Example

```
Dialog
CHECKBOX 1 "My Life as a Dog"
CHECKBOX 0 "A Boy's Life"
CHECKBOX 0 "A Boy and His Dog"
CHECKBOX 0 "All Dogs Go to Heaven"
BUTTON DEFAULT "OK" RESUME
BUTTON "Cancel" RESUME
DIALOG END
WAIT RESUME
IF Checkbox (3) = 1 and CheckBox (2) = 0
PERFORM CheckBox3
ELSE
CANCEL
DIALOG CANCEL
Return
```

---

## DIALOG CONTROL

---

### DIALOG CONTROL DialogIndex Control ControlNum Update

This command allows the you to alter the appearance and behavior of dialog boxes after they have been created. The DIALOG CONTROL command also updates the attributes of a previously defined dialog control.

Note: This command does not apply to IBM TN3270 emulations.

#### Arguments

##### DialogIndex

The optional **DialogIndex** argument is an integer (from 0 to 15) identifying a particular dialog box. If it is not included, DCS uses zero as the default value.

##### Control

The **Control** argument indicates which control to update and is specified by one of the following keywords:

BUTTON	GROUPBOX	LISTBOX	RADIOGROUP
CHECKBOX	ICON	MESSAGE	WIDEBUTTON
EDITTEXT	ICONBUTTON	RADIOBUTTON	

##### ControlNum

The **ControlNum** argument is an integer specifying the desired control; the first control of each type is specified by the integer one. To specify the first button or list box in a dialog box, assign the integer 1 (one) to the **ControlNum** argument following the keyword **BUTTON** or **LISTBOX**.

##### Update

The effect of the **Update** argument varies, depending upon which keyword is specified. The specific syntax for each keyword is given below. The **Control** and **ControlNum** arguments which normally precede these keywords have been omitted for clarity.



---

## DIALOG CONTROL, *continued*

---

Keyword	Description
BACKGROUND (rval, gval, bval)	The BACKGROUND keyword directs DCS to set the background color for icons in dialog boxes, the text of list boxes and messages, and the titles of group boxes. The rval, gval and bval values represent the level of red, green, and blue. The number for a color level can range from 0 to 255, where 255 represents the greatest saturation of the color.
DISABLE	The DISABLE keyword directs DCS to make the control inactive (it appears dimmed).
ENABLE	The ENABLE keyword directs DCS to make the control active (it appears highlighted).
HIDE	The HIDE keyword directs DCS to remove the specified control from view. A control should be hidden before invoking the MOVE keyword.
MOVE (x, y)	The MOVE keyword directs DCS to move the control to a new location in the dialog box. A control should be hidden before invoking the MOVE keyword. The coordinate set (x, y) specifies where in the dialog box to place the control. See the DIALOG command to determine the proper logical unit dimensions for your system. The coordinates (x, y) are related to the dialog box which contains the control. The top left corner of the dialog box is considered (0,0).
SETFOCUS	The SETFOCUS keyword directs DCS to select a control in a dialog box as a default control.
SHOW	The SHOW keyword directs DCS to make a hidden control visible.
TEXT (rval, gval, bval)	The TEXT keyword directs DCS to set the color of the text in list boxes, messages, and titles of group boxes. The rval, gval and bval values represent the level of red, green and blue, respectively. The color level is a number from 0 to 255, where 255 represents the greatest saturation of the color.

---

---

## DIALOG CONTROL, *continued*

---

### Example

In this example:

```
Dialog "Color Text"
EditText (,,80) "Name: "
Message (,,80) " "
Button Default "&OK" Resume
Dialog End

Dialog Control EditText 1 Text (0, 255, 0) ;Green Text

Wait Resume

Dialog Control Edittext 1 Background (0,0,255)
;Blue Background
Dialog Update Message 1 "Thanks!!"
Dialog Control Message 1 Text (255,0,0) ;Red Text
Wait Delay "6"
Dialog Cancel
```

the script draws a dialog box with an edit text field, a message (initially blank), and a button. The script makes the text of the edit text field green. When you click **OK**, the background of the edit text field changes to blue, and the message updates to display "Thanks!!" in red. After a delay of six seconds, the script removes the dialog box from the screen.

In this example:

```
Dialog Control Button 1 SetFocus
```

the command highlights the first button in the dialog.

---

## DIALOG CONTROL, *continued*

---

In this script segment:

```
#Boolean = True
DIALOG "Select"
EDITTEXT (,,140) "New File Name"
BUTTON (,,80) "Delete File"
BEGIN
#Boolean = False
PERFORM "DeleteFile"
RESUME
END
BUTTON DEFAULT "&Ok"
BEGIN
#Boolean = False
RESUME
END
Dialog End

WHILE #Boolean
BEGIN
DIALOG CONTROL BUTTON 1 SETFOCUS
WAIT DELAY "1"
DIALOG CONTROL EDITTEXT 1 SETFOCUS
WAIT DELAY "1"
DIALOG CONTROL BUTTON 2 SETFOCUS
WAIT DELAY "1"
End
```

DCS shifts the focus from one control to another until one of the buttons in the dialog box is selected.

---

## (DIALOG) DIMENSION

---

### DIMENSION Control Attribute Value

The (DIALOG) DIMENSION command specifies the desired position and size of a control. It is used when the optional coordinate set is not included. You may use this command if you have not specified the coordinate set of a control.



**Note:** Dialog boxes in the script language are governed in most cases by the operating system (Windows 95/98/NT). Therefore, settings which affect the display of dialog boxes in Windows also affect dialogs created with the DCS Script Language. For example, if display settings are set to Large Fonts, scripted dialog boxes use large fonts and are larger than dialog boxes displayed when using Small Fonts.

### Arguments

#### Control

The Control argument is specified by one of the following keywords:

BUTTON	GROUPBOX	LISTBOX	RADIOGROUP
CHECKBOX	ICON	MESSAGE	
EDITTEXT	ICONBUTTON	RADIOBUTTON	

#### Attribute

The Attribute argument is specified by one of the following keywords:

Keyword	Description
WIDTH, HEIGHT	These keywords define the desired width and height of the specified Control.
HTAB, VTAB	These keywords define the desired horizontal and vertical distance between specified Controls of the same type.
RIGHTMARGIN, LEFTMARGIN	These keywords locate a Control from the right or left margin of the dialog box.

#### Value

The Value argument is a numeric specifying the desired dimension, in logical units, of the specified Item. See the DIALOG command to determine the proper logical unit dimensions for your system.

#### Comments

The (DIALOG) DIMENSION command affects only controls in the dialog box you are defining. The dimensions are reset to the system defaults when a new DIALOG command is started.

---

## (DIALOG) EDITTEXT

---

### **EDITTEXT (x, y, w, h) Box Prompt Text LIMIT NumChars PASSWORD**

The (DIALOG) EDITTEXT dialog control command displays an edit text control (both prompt characters and a text box) in a dialog box.

#### **Arguments**

##### **(x, y, w, h)**

The optional coordinate set **(x, y, w, h)** specifies the top left corner (x, y), width (w), and height (h) of the edit text control (prompt and text box). The height (h) coordinate modifies only the height of the text box, not the prompt.

See the DIALOG command to determine the proper logical unit dimensions for your system. The (x, y) coordinates are relative to the dialog box which contains the edit text control. The top left corner of the dialog box is considered (0,0).

If you include the coordinate set, you can specify one to four coordinates. For example, (,40,) specifies the width only. If you do not include a coordinate set, DCS assigns a default size and placement.

##### **Box**

The optional **Box** argument is an integer specifying the width of the text box. If you do not include the **Box** argument, DCS assigns a default width.

##### **Prompt**

The **Prompt** argument is a string specifying the character or text which will appear to the left of the text box. If you do not want a prompt, specify a null string (“”).

##### **Text**

The optional **Text** argument is a string specifying the text DCS initially displays in the text box. If the **Text** argument is not included, an empty text box is created.

##### **LIMIT NumChars**

The optional **LIMIT** clause limits the number of characters DCS will initially display when it creates the text box, as well as limiting the number of characters you may enter in the text box. The **NumChars** argument is an integer specifying the number of characters. If you do not include the **LIMIT** clause, you may enter a maximum of 254 characters in the text box.

##### **PASSWORD**

The optional **PASSWORD** keyword directs DCS to display an asterisk for each character you enter into the edit text box, instead of displaying the entered text. This keyword allows you to protect sensitive information (like passwords or authorization codes).

If you want DCS to display asterisks when it initially creates the text box, as well as display asterisks after characters are entered in the text box, you must include both the **PASSWORD** keyword and the **Text** argument. DCS will initially include as many asterisks as the number of characters you place in the **Text** argument.

---

## (DIALOG) EDITTEXT, *continued*

---

### Comments

The Prompt argument of an edit text control is a static control. The text box of an edit text control is an active control.

The (DIALOG) EDITTEXT function allows you to gather the characters you enter into the text box.

You may include a maximum of 64 edit text controls in a single dialog box.

### Example

These commands:

```
DIALOG
EDITTEXT 100 "Prompt:" "Default Text"
BUTTON (165, 5, ,) DEFAULT "OK" RESUME
DIALOG END
WAIT RESUME
$Text = EDITTEXT (1)
DISPLAY $Text
DIALOG CANCEL
```

display a dialog box.

DCS's script language allows you to enter a maximum of 254 characters in the text box. The text box in this example is 100 dialog units wide. When you select the **OK** button or the **Resume** option on the **Script** menu, the (DIALOG) EDITTEXT function will copy the text from the edit text box into \$Text. The DISPLAY command then shows the contents of \$Text in the session window.

---

## (DIALOG) GROUPBOX

---

### GROUPBOX (x, y, w, h) Message

The (DIALOG) GROUPBOX command displays a group box control in a dialog box.

#### Arguments

##### (x, y, w, h)

The optional coordinate set (x, y, w, h) specifies the top left corner (x, y), width (w), and height (h) of the group box. See the DIALOG command to determine the proper logical unit dimensions for your system. The (x, y) coordinates are relative to the dialog box which contains the group box. The top left corner of the dialog box is (0,0).

If the coordinate set is included, one to four coordinates can be specified (for example, (,40,) specifies width only). If you do not include a coordinate set, DCS assigns a default size and placement.

##### Message

The Message argument is a string specifying the message to display in the top left corner of the group box.

#### Comments

A group box is a static control.

You may include a maximum of 255 messages in a single dialog box.

Although the coordinate set is optional, you should explicitly define the coordinates. Since a group box is independent of the controls it encloses, DCS cannot accurately pick a default size and placement for a group box.

To specify a group box with no embedded text, use a null string (“”) for the Message argument.

#### Example

```
Dialog
GROUPBOX (20, 12, 120, 80) "Which Do You Like?"
CHECKBOX (30,30) 1 "My Life as a Dog"
CHECKBOX (30,45) 0 "A Boy's Life"
CHECKBOX (30,60) 0 "A Boy and His Dog"
CHECKBOX (30,75) 0 "All Dogs Go to Heaven"
BUTTON (20, 100,,) DEFAULT "OK" RESUME
BUTTON (80, 100,,) "Cancel" RESUME
DIALOG END

WAIT RESUME
IF Checkbox (3) = 1 and CheckBox (2) = 0
    PERFORM CheckBox3
ELSE
    CANCEL

DIALOG CANCEL
```

---

## (DIALOG) ICON

---

### ICON (x, y, w, h) IconId

The (DIALOG) ICON command displays the specified icon control in a dialog box.

#### Arguments

##### (x, y, w, h)

The optional coordinate set (x, y, w, h) specifies the top left corner (x, y), width (w), and height (h) of the icon. Specifying a width and height for an icon will not actually change the size of the icon, but will restrict or enlarge the space allocated to the display of the icon.

See the DIALOG command to determine the proper logical unit dimensions for your system. The (x, y) coordinates are relative to the dialog box which contains the icon. The top left corner of the dialog box is at (0,0).

If the coordinate set is included, one to four coordinates can be specified (for example, (,40,) specifies width only). If you do not include a coordinate set, DCS assigns a default size and placement.

##### IconId

The IconId argument is specified by one of the following keywords:

CAUTION	NOTE	STOP
DYNACOMM	QUESTION	<i>"icon library resource ID"</i>



**Note:** You must enclose an icon ID within quotation marks.

#### Comments

An icon is a static control.

You may include a maximum of 255 icons in a single dialog box.

#### Example

```
DIALOG
ICON CAUTION
Icon DCS
Icon Note
Icon Stop
Button Default "OK" Resume
Button "Cancel" Resume
Dialog End
Wait Resume
Dialog Cancel
```

In this example, the script displays icons based on keywords supplied to the ICON command. These icons do not have any other function in the dialog other than to make the dialog more attractive.



---

## (DIALOG) ICONBUTTON

---

### ICONBUTTON (x, y, w, h) IconId Default Title Command

The (DIALOG) ICONBUTTON command displays an icon button control. An icon button control is a button in the shape of an icon.

#### Arguments

##### (x, y, w, h)

The optional coordinate set (x, y, w, h) specifies the top left corner (x, y), width (w), and height (h) of the icon button. Specifying a width and height for an icon button will not actually change the size of the icon button, but will restrict or enlarge the space allocated to the display of the icon button.

See the DIALOG command to determine the proper logical unit dimensions for your system. The (x, y) coordinates are relative to the dialog box which contains the icon button. The top left corner of the dialog box is at (0,0).

If the coordinate set is included, one to four coordinates can be specified (for example, (,40) specifies width only). If you do not include a coordinate set, DCS assigns a default size and placement.

##### IconId

The IconId argument is specified by one of the following keywords:

CAUTION	NOTE	STOP
DYNACOMM	QUESTION	<i>"icon library resource ID"</i>



**Note:** You must enclose an icon ID within quotation marks.

#### Default

The optional **Default** argument is specified by one of the following keywords:

Keyword	Action
DEFAULT	Causes the icon button to be automatically clicked when the [RETURN] key is pressed, unless another button is highlighted.
CANCEL	Causes the icon button to be automatically clicked when the [ESC] key is pressed, regardless of which button is highlighted.

A default button is displayed in the dialog box with a black border. You may specify only one DEFAULT button per dialog box.

#### Title

The optional **Title** argument is a string specifying the characters to display beneath the icon button. If a **Title** is not included, no title is displayed beneath the icon button.

#### Command

The optional **Command** argument specifies a logical command (either a single command or

---

## (DIALOG) ICONBUTTON, *continued*

---

a command block). Clicking the icon button executes the logical command. If a command is not included, clicking the icon button has no effect.

### Comments

An icon button is an active control.

You may include a maximum of 128 buttons in a single dialog box.

Icon buttons cannot be activated by accelerator keys or keyboard shortcuts.

### Example

```
DIALOG
ICONBUTTON CAUTION "Caution" RESUME
Iconbutton DCS "DCS" Resume
Iconbutton Note "Note" Resume
Iconbutton Stop "Stop" Resume
Button Default "OK" Resume
Button "Cancel" Resume
Dialog End

Wait Resume

Dialog Cancel
```

This example shows icon buttons which, in addition to providing an attractive quality as shown in the `ICON` command example, also function like any dialog button. The icons displayed in this script are referred to by their keywords.

```
DIALOG
Iconbutton "_BinXfers" "Transfers" Resume
Iconbutton DCS "DCS" Resume
Iconbutton Note "Note" Resume
Iconbutton Stop "Stop" Resume
Button Default "OK" Resume
Button "Cancel" Resume
Dialog End

Wait Resume

Dialog Cancel
```

This example is similar to the previous example. However, the icon for the first icon button is retrieved from DCS's icon library, `FSEICONS.DLL`.

---

## (DIALOG) LISTBOX

---

### **LISTBOX (x, y, w, h) Table Record INVERT COMBOBOX Command**

The (DIALOG) LISTBOX command displays a list box control in a dialog box.

#### **Arguments**

##### **(x, y, w, h)**

The optional coordinate set **(x, y, w, h)** specifies the top left corner (x, y), width (w), and height (h) of the list box. See the **DIALOG** command to determine the proper logical unit dimensions for your system. The (x, y) coordinates are relative to the dialog box which contains the list box. The top left corner of the dialog box is at (0,0).

If the coordinate set is included, one to four coordinates can be specified (for example, (,40,) specifies width only). If you do not include a coordinate set, DCS assigns a default size and placement.

##### **Table**

The **Table** argument is an integer (from 0 to 15) identifying the table containing the desired data.

##### **Record**

The optional **Record** argument is an integer (from 0 to n) specifying which record number DCS initially selects. If no **Record** argument is included, no record is initially selected. You may specify a record number of -1, or use the **DIALOG UPDATE** command to ensure that no record is selected. It is best to put parentheses around the number to prevent the compiler from misinterpreting the expression (for example, table = table (-1)).

##### **INVERT**

The optional **INVERT** keyword lists the records in reverse order.

##### **Command**

The optional **Command** argument specifies a logical command (either a single command or a command block). Selecting an item in the list box executes the logical command. If a command is not included, selecting a list box item has no effect.

##### **COMBOBOX**

The optional **COMBOBOX** parameter reconstructs the list box into a drop-down list of items. DCS displays the selected item in the combo box. The other items appear when the down arrow is selected. The coordinates used for this type of list box must specify the size of the fully open box.

---

## (DIALOG) LISTBOX, *continued*

---

### Comments

If a list box in a dialog box contains only columns of data, associate the structured table with the list box and use the `SYSTEM` command with the `0x0803` parameter to force the text in the list box to be displayed in a fixed-width font.

A list box is an active control.

If the list box item is selected by double-clicking the mouse, the default button (if included) is also clicked.

You can obtain the number of the selected record using the `(DIALOG) LISTBOX` function.

You may include a maximum of ten list boxes in a single dialog box.

### Example

```
Table Define 0 Fields Char 12 File
TABLE LOAD 0 FROM "C:\DCSERIES\Memo\Table.TXT" AS TEXT
Dialog
Listbox 0 5 Invert
Button Default "OK" Resume
Button "Cancel" Resume
Dialog End

Wait Resume

Record Read 0 At Listbox ()
Display @R0 | ""^M"
Dialog Cancel
Table Close 0
```

The data stored in the file named `TABLE . TXT` is loaded into Table 0 (zero) and then displayed (in reverse order) in the list box.

---

## (DIALOG) MESSAGE

---

### MESSAGE (x, y, w, h) Text

The (DIALOG) MESSAGE command displays a message control in a dialog box.

#### Arguments

##### (x, y, w, h)

The optional coordinate set (x, y, w, h) specifies the top left corner (x, y), width (w), and height (h) of the message. Specifying a width and height for a message will not actually change the size of the message, but will restrict or enlarge the space allocated to the display of the message.

See the DIALOG command to determine the proper logical unit dimensions for your system. the (x, y) coordinates are relative to the dialog box which contains the message. The top left corner of the dialog box is (0,0).

If the coordinate set is included, one to four coordinates can be specified (for example, (,40,) specifies width only). If you do not include a coordinate set, DCS assigns a default size and placement.

##### Text

The Text argument is a string or string variable specifying the message to display.

#### Comments

A message is a static control.

You may include a maximum of 255 messages in a single dialog box.

#### Example

```
Table Define 0 Fields Char 12 File
TABLE LOAD 0 FROM "C:\DCSERIES\Memo\Table.TXT" AS TEXT
Dialog
Message "Select a record from the file:"
Listbox 0 5 Invert
Button Default "OK" Resume
Button "Cancel" Resume
Dialog End
Wait Resume
Record Read 0 At Listbox ()
Dialog Cancel
;Manipulate the record
;
;
Table Close 0
```

---

## (DIALOG) NEWLINE

---

### NEWLINE

The (DIALOG) NEWLINE command places the next specified dialog box control on the next line of the dialog box.

### Arguments

The (DIALOG) NEWLINE command takes no arguments.

### Comments

This command may be used to place dialog box controls on successive lines, without using the optional coordinate set. (DIALOG) NEWLINE controls vertical placement only.

### Example

In this example:

```
DIALOG
MESSAGE "The time is: " | TIME ()
NEWLINE
MESSAGE "The date is: " | DATE ()
NEWLINE
NEWLINE
BUTTON "OK" RESUME
DIALOG END
WAIT RESUME
DIALOG CANCEL
```

the (DIALOG) NEWLINE command places the date message one line down from the time message, and places the **OK** button two lines down from the date message.

---

## (DIALOG) PICTURE

---

### **PICTURE (x, y, w, h) PictureId**

The (DIALOG) PICTURE dialog control command displays a picture in a dialog box.

#### **Arguments**

##### **(x, y, w, h)**

The optional coordinate set (x, y, w, h) specifies the top left corner (x, y), width (w), and height (h) of the picture. Specifying a width and height for a picture will not actually change the size of the picture, but will restrict or enlarge the space allocated to the display of the picture.

See the DIALOG command to determine the proper logical unit dimensions for your system. The (x, y) coordinates are relative to the dialog box which contains the picture. The top left corner of the dialog box is (0,0).

If the coordinate set is included, one to four coordinates can be specified (for example, (,40) specifies width only). If you do not include a coordinate set, DCS assigns a default size and placement.

##### **PictureId**

The PictureId argument is a string specifying the path and file name of a .bmp file (a device independent bitmap format). You must include the proper extension of the file format.

#### **Comments**

A picture is a static control. A dialog box can display only one picture.

#### **Example**

```
Dialog (, , 77, 145)
PICTURE (5, 5, , ) "C:\Graphics\Shuttle.BMP"
Button (2, 125, , ) Default "OK" Resume
Button (43, 125, , ) "Cancel" Resume
Dialog End

Wait Resume
Dialog Cancel
Return
```

---

## (DIALOG) RADIOBUTTON

---

### **RADIOBUTTON (x, y, w, h) ButtonName**

The (DIALOG) RADIOBUTTON command displays a radio button control in a dialog box.

#### **Arguments**

##### **(x, y, w, h)**

The optional coordinate set **(x, y, w, h)** specifies the top left corner (x, y), width (w), and height (h) of the radio button. Specifying a width and height for a radio button will not actually change the size of the radio button, but will restrict or enlarge the space allocated to display the radio button.

See the DIALOG command to determine the proper logical unit dimensions for your system. The (x, y) coordinates are relative to the dialog box which contains the radio button. The top left corner of the dialog box is (0,0).

If the coordinate set is included, one to four coordinates can be specified (for example, (,40,) specifies width only). If you do not include a coordinate set, DCS assigns a default size and placement.

##### **ButtonName**

The **ButtonName** argument is a string describing the radio button option. It is displayed to the right of the radio button.

If any character in the **ButtonName** argument is preceded by an ampersand (&), the character will appear underlined in the radio button name. The radio button can then be selected by pressing the [ALT] key and the key of the underlined character simultaneously.

#### **Comments**

A radio button is an active control.

A (DIALOG) RADIOBUTTON command must follow a (DIALOG) RADIOBUTTON command. You may include a maximum of 255 radio buttons in a single dialog box.

#### **Example**

See the (DIALOG) RADIOGROUP command.



---

## (DIALOG) RADIOGROUP

---

### **RADIOGROUP (x, y, w, h) Default GroupName Command**

The (DIALOG) RADIOGROUP command defines a group of radio buttons in a dialog box.

#### **Arguments**

##### **(x, y, w, h)**

The optional coordinate set (x, y, w, h) specifies the top left corner (x, y), width (w), and height (h) of the radio group title. Specifying a width and height for a radio button will not actually change the size of the radio button title, but will restrict or enlarge the space allocated to display the radio button title.

See the DIALOG command to determine the proper logical unit dimensions for your system. The (x, y) coordinates are relative to the dialog box which contains the radio buttons. The top left corner of the dialog box is (0,0).

If the coordinate set is included, one to four coordinates can be specified (for example, (,40,) specifies width only). If you do not include a coordinate set, DCS assigns a default size and placement.

##### **Default**

The optional **Default** argument is an integer specifying which radio button DCS initially selects. If the **Default** argument is not included, no radio button is initially selected.

##### **GroupName**

The **GroupName** argument is a string specifying the title of the radio group, which is displayed above the radio group.

If any character in the **GroupName** argument is preceded by an ampersand (&), the character appears underlined in the radio group title. Pressing the [ALT] key and the key of the underlined character simultaneously highlights the nearest active control.

##### **Command**

The optional **Command** argument specifies a logical command (either a single command or a command block). Selecting a radio button executes the logical command. If a command is not included, selecting a radio button has no effect.

#### **Comments**

A radio group is a static control.

Radio buttons are numbered sequentially (from 1 to *n*) within a radio group. The radio buttons are numbered according to the order in which they appear in the script, not according to their placement on the screen.

You can obtain the number of the radio button selected using the (DIALOG) RADIOGROUP function.

You may include a maximum of 16 radio groups in a single dialog box.

---

## (DIALOG) RADIOGROUP, *continued*

---

### Example

```
Dialog
RadioGroup (30,15) "Select One"
RadioButton (30,30) "My Life as a Dog"
RadioButton (30,45) "A Boy's Life"
RadioButton (30,60) "A Boy and His Dog"
RadioButton (30,75) "All Dogs Go to Heaven"
Button (20, 100,,) Default "OK" Resume
Button (80, 100,,) "Cancel" Resume
Dialog End

Wait Resume
If RadioGroup (1) = 1
Perform RadioGroup1
Else
Cancel

Dialog Cancel

DIALOG (, , 250, )
RADIOGROUP (30,15) "Select One Here"
RADIOBUTTON (30,30) "My Life as a Dog"
RADIOBUTTON (30,45) "A Boy's Life"
RADIOBUTTON (30,60) "A Boy and His Dog"
RADIOBUTTON (30,75) "All Dogs Go to Heaven"

RADIOGROUP (140,15) "Select One Here Also"
RADIOBUTTON (140,30) "Goin' South"
RADIOBUTTON (140,45) "Gone with the Wind"
RADIOBUTTON (140,60) "The Wind and the Lion"
RADIOBUTTON (140,75) "A Lion in Winter"

BUTTON (20, 100,,) Default "OK" RESUME
BUTTON (80, 100,,) "Cancel" RESUME
DIALOG END

WAIT RESUME

IF (RadioGroup (1) = 3) and (RadioGroup (2) = 1)
PERFORM SubCrazy
ELSE
CANCEL

DIALOG CANCEL

RETURN
```

---

## DIALOG UPDATE

---

### DIALOG UPDATE DialogIndex Control ControlNum Update

The DIALOG UPDATE command updates a previously defined dialog control.

#### Arguments

##### DialogIndex

The optional **DialogIndex** argument is an integer (from 0 to 15) identifying a particular dialog box. If it is not included, DCS uses zero as the default value.

##### Control

The **Control** argument indicates which control to update and is specified by one of the following keywords:

BUTTON	GROUPBOX	LISTBOX	RADIOBUTTON
CHECKBOX	ICON	MESSAGE	RADIOGROUP
EDITTEXT	ICONBUTTON	PICTURE	WIDEBUTTON

##### ControlNum

The **ControlNum** argument is an integer specifying the control; the first control of each type is specified by the integer 1 (one).

The **ControlNum** argument must follow each **Control** argument keyword, except the **PICTURE** keyword.

##### Update

The effect of the **Update** argument varies, depending upon which control attribute keywords is specified. The syntax for each keyword is given in the following text. The **ControlNum** argument has been omitted for clarity.

Keyword	Action
BUTTON Title	Directs DCS to update the title of the specified button. The Title argument is a string specifying the updated button title.
CHECKBOX Default Text	Directs DCS to update the state of the check box and update the text displayed to the right of the specified check box. You must specify either the Default or the Text argument, or both. The Default argument is an integer (either zero or one) specifying a new default status of the check box; zero specifies unchecked, and one specifies checked. The Text argument is a string specifying the new text to be displayed to the right of the check box.
EDITTEXT TEXT	Updates the contents of the specified edit text box using the Text argument.
GROUPBOX Message	Updates the embedded text of the specified group box. The Message argument is a string specifying the replacement for the current message string.

## DIALOG UPDATE, *continued*

Keyword	Action
ICON IconId	Updates the specified icon. The IconId argument consists of one of the available icon keywords, or a string specifying the ID of the new icon to be displayed.
ICONBUTTON IconId Title	<p>Updates the title and the icon of the specified icon button. The ICONBUTTON keyword requires one or both of the following arguments:</p> <ul style="list-style-type: none"> <li>▼ IconId An icon keyword or a string that specifies the ID of the new icon; and</li> <li>▼ Title A string that specifies the new text to appear beneath the icon button.</li> </ul>
LISTBOX	<p>Updates the data in the specified list box. You may specify either a new table to be loaded, or a modification in a single record.</p> <p>New Table:</p> <p>TABLE number <b>Record</b> <b>INVERT</b></p> <p>If a new table is to be loaded, the LISTBOX keyword may have three arguments:</p> <ul style="list-style-type: none"> <li>▼ The first keyword, TABLE, is followed by the number of the table containing the new data.</li> <li>▼ The optional second keyword, <b>Record</b>, is an integer specifying the record number to be selected by default.</li> <li>▼ The optional third keyword, <b>INVERT</b>, which directs DCS to display the records in reverse order.</li> </ul> <p>Modify Record:</p> <p><b>Record Text</b></p> <p>If a single record is to be modified, the LISTBOX keyword may have one or two arguments:</p> <ul style="list-style-type: none"> <li>▼ The optional first keyword, <b>Record</b>, is an optional integer specifying which record to modify. If <b>Record</b> is not included, the currently selected record is modified.</li> <li>▼ The second keyword, <b>Text</b>, is a string specifying the replacement data for the specified record.</li> </ul>
MESSAGE Text	Updates the specified message. The Text argument is a string replacing the current message string.
PICTURE FileName	Changes the BMP or WMF image currently displayed in the a dialog box. Do not include the ControlNum argument with the PICTURE keyword. FileName specifies the path and name of the picture.
RADIOBUTTON ButtonName	Updates the text displayed to the right of the specified radio button. The ButtonName argument is a string specifying the new text to appear to the right of the radio button.

---

## DIALOG UPDATE, *continued*

---

Keyword	Action
RADIOGROUP Default GroupName	Updates the default radio button and the title of the specified radio group. The RADIOGROUP keyword requires one or both of the following arguments: <ul style="list-style-type: none"><li>▼ Default, an integer specifying which radio button within the radio group is to be selected initially; and</li><li>▼ GroupName, a string specifying the new title to appear above the radio group.</li></ul>
WIDEBUTTON Title	Updates the title of the specified wide button. The Title argument is a string specifying the new title.

---

### Example

In this example:

```
DIALOG (,,260,100) "GIFT LIST"
MESSAGE (,,400) "Merry Christmas!"
EDITTEXT 100 "Enter Name:"
NEWLINE
EDITTEXT 120 "Enter Desired Gift:"
BUTTON "Enter" PERFORM update
BUTTON "Exit" RESUME
DIALOG END
WAIT RESUME
DIALOG CANCEL
CANCEL

*update
DIALOG UPDATE MESSAGE 1 "Thanks " | edittext(1)
RETURN
```

when the Enter button is clicked, the message "Merry Christmas!" is updated to display "Thanks" followed by the contents of edittext1.

---

# DISCONNECT

---

## DISCONNECT WINDOW WinHandle

The DISCONNECT command terminates the connection with the host.

### Arguments

#### WINDOW WinHandle

The optional **WINDOW** clause includes the WINDOW keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

The **WINDOW** clause terminates the session in the window specified by the **WinHandle** argument. If the **WINDOW** clause is not included, all connected sessions will be terminated.

### Example

In this example:

```
LOAD "HOST7"  
CONNECT  
IF ERROR () or NOT CONNECT ()  
CANCEL  
FILE SEND BINARY "DATA.TXT" "" "secret" AS "data1"  
IF ERROR ()  
DISPLAY "File not sent"  
DISCONNECT
```

DCS loads a session file and connects to the host. After the file transfer is complete, DCS disconnects from the host, terminating the session.

---

## DISPLAY

---

### DISPLAY (Row, Col) String CRONLY WINDOW WinHandle

The DISPLAY command writes a sequence of characters into a connected session window, but it does not send the text to the remote system.

#### Arguments

##### (Row, Col)

The optional (Row, Col) coordinate set indicates the row and column in the session window in which to display the string, where the first row is row zero, and the first column is column zero.

If you do not include coordinates, DCS displays the string starting at the current cursor position.


If you provide a coordinate which is larger than the number of rows or columns in the session window (e.g., 75, 75), the string is displayed either in the status bar (the area between the session window and the toolbar or bottom of the DCS application window) or at position 0,0. The display location is dependent on the particular emulation.

##### String

The String argument contains the characters to display.

##### CRONLY

The optional CRONLY keyword causes the DISPLAY command to display carriage returns as carriage returns only (not followed by line feeds). If it is not included, the DISPLAY command displays a line feed character after every carriage return character displayed in the string.

 **Note:** The CRONLY keyword is not applicable to 3270 emulations.

##### WINDOW WinHandle

The optional WINDOW clause includes the WINDOW keyword and the WinHandle argument. The WinHandle argument is an integer identifying a particular session window.

If a session is connected in a session window, the WINDOW clause causes DCS to display the String argument in the session window specified by the WinHandle argument. If the WINDOW clause is not included, DCS displays the contents of the String argument in the currently highlighted and connected session window.

#### Comments

The DISPLAY command only displays text in a session which is connected to a host.

The DISPLAY command can help to debug a script. If errors occur, strategically placed DISPLAY commands might pinpoint which command is generating the error (see the second example).

The DISPLAY command displays a maximum of 254 characters at a time. For example, consider that if \$String1 and \$String2 have 254 characters each, the following script lines

---

## DISPLAY, *continued*

---

will only display \$String1:

```
DISPLAY $String1 | "^M"  
;DCS will not display the Control-M, a  
;carriage return, after displaying $String1 (the  
;terminal cursor will remain at the end of $String1  
;in the session window), since $String1 has 254  
;characters  
DISPLAY $String1 | $String2  
;DCS will not display $String2 (the terminal  
;cursor will be in the same place as in the script  
;line above), since $String1 has 254 characters
```

### Example

In this example:

```
TABLE DEFINE 0 FIELDS CHAR 10 INT 5 INT 5  
TABLE LOAD 0 FROM "BUDGET.XLS" AS SYLK  
SET %1 0  
WHILE NOT EOF ()  
BEGIN  
RECORD READ 0  
DISPLAY (%1,0) @R0  
INCREMENT %1  
END
```

and assuming a connected session, DCS defines and loads an Excel spreadsheet into Table 0 (zero). DCS reads each record and allows you to examine them in the session window using the DISPLAY command.

In this example:

```
TABLE DEFINE 0 FIELDS CHAR 20 CHAR 40  
IF ERROR ()  
BEGIN  
DISPLAY "Table 0 not defined"  
BEEP1  
END  
TABLE DEFINE 1 TEXT "MYFILE"  
IF ERROR ()  
BEGIN  
DISPLAY "Table 1 not defined"  
BEEP1  
END  
TABLE LOAD 0 FROM "DATA" AS SYLK  
IF ERROR ()  
BEGIN  
DISPLAY "Table 0 not loaded"  
BEEP1  
END
```

and assuming a connected session, the DISPLAY command is used as a debugging tool to help determine which command is not executing as expected. If an error occurs, the messages displayed in the session window might point to the problem.



---

# DISPLAYCONFIG

---

## DISPLAYCONFIG String WINDOW WinHandle

The DISPLAYCONFIG command sets display parameters for a session window. These settings are available on the **Displays** tab of the **Session Properties** dialog.

### Arguments

#### String

The String argument is composed of a keyword followed by the assignment operator (=) and a valid setting.

The keywords and their values are in the table below:

General Tab Keywords	Type	Value(s)
TopStatusLine	Boolean	1 (true), 0 (false)
BottomStatusLine	Boolean	1 (true), 0 (false)
HorizontalScrollbarMode	String	0 (auto), 1 (off), 2 (on)
VerticalScrollbarMode	String	0 (auto), 1 (off), 2 (on)
ShowSessionButtons	Boolean	1 (true), 0 (false)
TerminalBorder	Boolean	1 (true), 0 (false)
MaximumScrollRate	Integer	1, 2, 3, 4, 5

Font Tab Keywords	Type	Value(s)
AutoSizeFont	Boolean	1 (true), 0 (false)
FontSize	String	w, h, where w is an integer indicating the width in pixels of the font, and where h is an integer indicating the height in pixels of the font.
MinFontSize	String	w,h, where w is an integer indicating the width in pixels of the font, and where h is an integer indicating the height in pixels of the font.
UseEmulationFont	Boolean	1 (true), 0 (false)
FontFaceName	String	The name of a monospace or fixed pitch font on your system.
Blinking	Boolean	1 (true), 0 (false)

Cursor Tab Keywords	Type	Value(s)
CursorVisible	String	On, Off
CursorType	String	Block
CursorBlink	String	Blink, Steady
AutoScrollToCursor	Boolean	1 (true), 0 (false)
HistorySize	Integer	0 to 9999
ReverseHistory	Boolean	1 (true), 0 (false)
SaveClearedScreen	Boolean	1 (true), 0 (false)

---

## DISPLAYCONFIG, *continued*

---

### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The optional **WinHandle** argument is an integer identifying a particular child window.

### Comments

The **ERROR** function returns **TRUE** if the **WinHandle** or **String** keyword is invalid.

 Also see: **GETDISPLAYCONFIG** command

### Example

These script lines:

```
DisplayConfig "TerminalBorder=True"  
DisplayConfig "MaximumScrollRate=16"  
DisplayConfig "AutoSizeFont=True"
```

affect the active session window. They turn on the terminal border, sets the scroll rate to the highest setting, and require DCS to automatically resize fonts as the session window is resized.

These script lines:

```
DisplayConfig "CursorType=On"  
DisplayConfig "CursorType=Block"
```

affect the currently highlighted session window. They turn on the session cursor as a block character in the session window.

---

# DROPDTR

---

## DROPDTR DelayUnits

The DROPDTR command directs DCS to hold low the DTR line of a session's serial port for a specified number of milliseconds.



**Note:** This command does not apply to IBM TN3270 emulations.

### Arguments

#### DelayUnits

The DelayUnits argument is an integer specifying the length of time the DTR line is to be held low, where one delay unit equals one millisecond.

### Comments

This command is valid only for the serial ports and modems.

### Example

In this example:

```
*hangup
SEND "logout"
DROPDTR 300
```

DCS sends the logout command to the remote system, then drops the DTR line low for 300 milliseconds in order to break the connection.

---

## EDIT COPY

---

### EDIT COPY String WINDOW WinHandle

The EDIT COPY command copies characters to the clipboard. Executing the EDIT COPY command is equivalent to selecting **Copy** on the **Edit** menu.

#### Arguments

##### String

The optional **String** argument specifies text to copy to the clipboard. If you have not included the **String** argument, DCS will copy the visible selection made in the currently active child window. If you have not included the **String** argument and no characters are selected in a DCS child window, DCS will not affect the clipboard contents.

##### WINDOW WinHandle

The optional **WINDOW** clause is composed of both the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular child window. The **WINDOW** clause specifies the current selection in the window identified by the **WinHandle** argument.

#### Comments

If the **WINDOW** clause and the **String** argument are not included in the command, the current selection in the active edit window is copied. However, if you include both the **WINDOW** clause and the **String** argument in the command, the command will default to copying the **String** argument only.

The EDIT COPY command does not work in concert with the SELECTION or SELECTION BUFFER commands (invisible selections), but works with visible selections you make with your mouse or cursor keys.

#### Example

In this example:

```
EDIT COPY SCREEN (0, 0, ,%WnHnd)
```

the SCREEN function provides the first line of a window (whose window handle is contained in the variable %WnHnd) to the EDIT COPY command. The EDIT COPY command then places a copy of the text into the clipboard.

This example:

```
EDIT COPY Window %Session1Hnd
```

copies any text selected in the session window specified by the window handle contained in %Session1Hnd.

---

## EDIT COPYSPECIAL

---

### EDIT COPYSPECIAL Destination Format WINDOW WinHandle

The EDIT COPYSPECIAL command copies text to a destination.



Note: The EDIT COPYSPECIAL command does not apply to the IBM TN3270 emulation.

#### Arguments

##### Destination

The Destination argument is specified by one of the following keywords:

Keyword	Action
PRINTER	<p>If you want to send text from a session window to the printer, you must use the PRINTER keyword as the Destination argument and the TEXT keyword as the Format argument (see the Format argument below). The PRINTER keyword assumes that the script has made a selection in a session window with the SELECTION command and also assumes that the session window is the active window.</p> <p>If you want to send a graphical capture of a session to the printer, you must include the BITMAP keyword as the Format argument (see the Format argument below).</p> <p>The PRINTER keyword assumes that a session window is active. The command captures the entire DCS application window and sends it to the printer (regardless of whether the script has made a selection in the session window).</p>
FILE FileName	<p>If you want to save text from a session window to a file, you must use the FILE phrase as the Destination argument and the TEXT keyword as the Format argument (see the Format argument below). The FILE phrase is composed of the keyword FILE and an optional string argument, <b>FileName</b>. The FILE phrase assumes that the script has made a selection in the session window with the SELECTION command and also assumes that the session window is the active window.</p>
CLIPBOARD	<p>The CLIPBOARD keyword is valid with all of the formats allowed for this command (see the Format argument below). When you include the TEXT or TABLE keywords, the CLIPBOARD keyword makes the same assumptions as the PRINTER keyword, but copies the text to the clipboard.</p> <p>If you want to copy a graphical capture of a session to the clipboard, you must include the BITMAP keyword as the Format argument (see the Format argument below). Once copied to the clipboard, the image may be pasted it into an open file in an application which supports the bitmap format (such as Windows Paint).</p> <p>If a session window is active, this command captures the entire DCS application window and places it in the clipboard (regardless of whether the script has made a selection in the session window).</p>

---

## EDIT COPYSPECIAL, *continued*

---

### Format

The Format argument may be one of the following keywords:

Keyword	Availability
TEXT	Available for all three destination keywords (see the Destination argument above).
BITMAP	Available only with the CLIPBOARD and PRINTER keywords.
TABLE	Available only with the CLIPBOARD keyword.

### WINDOW WinHandle

The WINDOW clause and WinHandle argument must be included. The window handle is an integer specifying a session window.

### Comments

The ERROR function is set to TRUE if DCS cannot complete this command.

When copying text, the EDIT COPYSPECIAL command works in concert with the SELECTION or SELECTION BUFFER commands.

---

## EDIT COPYSPECIAL, *continued*

---

### Example

This script:

```
$Set = "Main"
%ChrNHnd = 6
$WHnd = HWndList ()
#Found = False
%Ptr = 1
WHILE NOT #Found AND (%Ptr < Length ($WHnd))
BEGIN
IF WndTitle(Num ((Substr ($WHnd, %Ptr, %ChrNHnd)))= \
$Set
BEGIN
#Found = True
%TermWinHnd = Num (Substr ($WHnd, %Ptr, %ChrNHnd))
END
Else
BEGIN
%Ptr = %Ptr + %ChrNHnd + 1
END
END

WINDOW ACTIVATE %TermWinHnd
SELECTION 0 5
EDIT COPYSPECIAL FILE "Ed1.TXT" TEXT WINDOW %TermWinHnd
WINDOW OPEN MEMO "Ed1.TXT"
Cancel
```

assumes the active session is `Main` and uses the `HWNDLIST` function to retrieve a list of window handles of the open DCS child windows. Next in the `WHILE` loop, the script searches for the window handle of the session window. When the script finds the handle, it activates the session window, selects the first six visible lines in the window, copies the lines to a memo window, and then opens the memo window.

---

## EDIT CUT

---

### EDIT CUT WINDOW WinHandle

The EDIT CUT command copies the current selection from the active child window to the clipboard and then deletes the selection from the window.

#### Arguments

##### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The optional **WinHandle** argument is an integer identifying a particular child window.

The **WINDOW** clause specifies the current selection in the window identified by the **WinHandle** argument.

#### Comments

If the **WINDOW** clause is not included in the command, the current selection in the active edit window is copied, and then deleted, from the window. If the **WinHandle** argument is the handle of a session window, the selection is only copied to the clipboard; the selection is not deleted from the session window.

The EDIT CUT command does not work in concert with the SELECTION or SELECTION BUFFER commands, but does work with selections you make with your cursor keys or mouse.

In script and memo windows, executing the EDIT CUT command is equivalent to selecting **Cut** on the **Edit** menu.

#### Example

In this example:

```
EDIT FIND "version 4.70"  
EDIT CUT
```

DCS searches the active document window for the string version 4.70 and cuts the selection from the document.



---

## EDIT FIND

---

### EDIT FIND String CASE REVERSE WINDOW WinHandle

The EDIT FIND command searches a child window for a string and, if found, selects it.

#### Arguments

##### String

The String argument specifies the string for which to search.

##### CASE

The optional CASE keyword directs DCS to perform a case-sensitive search. If it is not included, the search is performed without regard to the capitalization of the specified string.

##### REVERSE

The optional REVERSE keyword directs DCS to search the text backward from the current document position. If it is not included, the search is performed forward from the current position.

##### WINDOW WinHandle

The optional WINDOW clause includes the WINDOW keyword and the WinHandle argument. The WinHandle argument is an integer identifying a particular child window.

The WINDOW clause directs DCS to search a window specified by the WinHandle argument.

#### Comments

If the WINDOW clause is not included in the command, DCS will search the active edit window.

If DCS cannot find the contents of String in the window, the ERROR function returns TRUE.

#### Example

In this example:

```
EDIT FIND "DCS version 4.0"  
IF ERROR ()  
  DISPLAY "Old version"  
ELSE  
  DISPLAY "New version"
```

DCS searches the document window for the specified string. If it is found, "New version" is displayed in the session window. If it is not found, "Old version" is displayed.

---

## EDIT GOTO

---

### EDIT GOTO Line WINDOW WinHandle

The EDIT GOTO command positions the cursor at the beginning of the specified line in a child window.

#### Arguments

##### Line

The **Line** argument is an integer specifying the line number of the cursor.

##### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument.

The **WinHandle** argument is an integer identifying a particular child window.

The **WINDOW** clause directs DCS to position the cursor in a window specified by the **WinHandle** argument.

#### Comments

If the **WINDOW** clause is not included in the command, DCS will position the cursor in the active edit window. If a **Line** argument larger than the number of lines in the document is specified, DCS will position the cursor at the last line of the document.

#### Example

In this example:

```
%line = 1
WHILE %line <= 20
BEGIN
EDIT GOTO %line
EDIT COPY STR (%line) | " "
EDIT PASTE
INCREMENT %line
END
```

DCS places the cursor at the beginning of each line as it numbers the first 20 lines in the document window.

---

## EDIT PASTE

---

### EDIT PASTE WINDOW WinHandle

The EDIT PASTE command copies text from the clipboard to the current selection in a DCS child window.

#### Arguments

##### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular child window.

The **WINDOW** clause directs DCS to paste the contents of the clipboard as a text selection in a window specified by the **WinHandle** argument.

#### Comments

If the **WINDOW** clause is not included in the command, DCS will paste the text into the active edit window.

In edit windows, executing the EDIT PASTE command is equivalent to selecting **Paste** on the **Edit** menu.

#### Example

In this example:

```
%line = 1
WHILE %line <= 20
BEGIN
EDIT GOTO %line
EDIT COPY STR (%line) | " "
EDIT PASTE
INCREMENT %line
END
```

the first 20 lines in the document window are numbered. As the value of `%line` increases, the contents of `%line` are copied to the clipboard and then pasted into the edit window.

---

## EDIT REPLACE

---

### EDIT REPLACE **String1 String2 CASE REVERSE WINDOW WinHandle**

The EDIT REPLACE command searches a script or memo window for **String1** and replaces it with **String2**.

#### Arguments

##### **String1**

The **String1** argument specifies the string for which to search.

##### **String2**

The **String2** argument specifies the text which is to replace the indicated string.

##### **CASE**

The optional **CASE** keyword performs a case-sensitive search. If it is not included, the search is performed without regard to the capitalization of the specified string.

##### **REVERSE**

The optional **REVERSE** keyword searches the text backward from the current document position. If it is not included, the search is performed forward from the current position.

##### **WINDOW WinHandle**

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular child window.

The **WINDOW** clause replaces **String1** in a window, which is specified by the **WinHandle** argument.

#### Comments

If the **WINDOW** clause is not included in the command, DCS will replace **String1** in the active script or memo window.

In script or memo windows, executing the EDIT REPLACE command is equivalent to selecting **Replace** on the **Edit** menu.

Each execution of the EDIT REPLACE command searches for and replaces only the first instance of **String1** that it finds.

The **ERROR** function returns **TRUE** if **String1** is not found.

---

## EDIT REPLACE, *continued*

---

### Example

These commands:

```
EDIT GOTO 1
WHILE NOT ERROR ()
EDIT REPLACE "cat" "dog"
```

establish a loop whereby each instance of the string "cat" is replaced by the word "dog" in the active edit window.

---

# EMULCONFIG

---


## EMULCONFIG String WINDOW WinHandle

The EMULCONFIG command is used to set the value of a parameter used in the emulator configuration for the active session.

### Arguments

#### String

The String argument includes a keyword followed by the assignment operator (=) and a valid setting. Together, the keyword and the setting are used to configure the emulation for the active session.

 **Note:** Configuration keywords for emulations shipped with client options, such as Tandem 6530 and TN3270, are valid only if the client option has been installed.

The first six tables appearing below and on the next two pages list keywords and valid settings for the emulations provided in the base product. The remaining tables list keywords and valid settings for individual client options that are purchased installed separately.

Keywords for ADDS Viewpoint	Valid Setting(s)
LOCALECHO	1 (true), 0 (false)
AUTOSCROLL	1 (true), 0 (false)
AUTOLINEFEED	1 (true), 0 (false)
LINETERMINATOR	none, cr, creat, cretx

Keywords for ANSI	Valid Setting(s)
MODE	ansi, tty, scoansi
LINEWRAP	1 (true), 0 (false)
DESTBACKSPACE	1 (true), 0 (false)
LOCALECHO	1 (true), 0 (false)
CODEPAGE850	1 (true), 0 (false)
SCREENROW	24, 25

---

## EMULCONFIG, *continued*

---

<b>Keywords for AT&amp;T 605/705</b>	<b>Valid Setting(s)</b>
MODE	att605, att705, pcxt, pcat1, pcat2
BITCONTROL	7, 8
SWAP DELETE	1 (true), 0 (false)
SCREENROW	24, 25
NEWLINE	1 (true), 0 (false)
LOCALECHO	1 (true), 0 (false)
DESTRUCTIVEBS	1 (true), 0 (false)
COLUMNS	80, 132
LINEWRAP	1 (true), 0 (false)
NUMPADAPPLICATION	1 (true), 0 (false)
LIGHTSCREEN	(true), 0 (false)
DISPLAYCONTROLS	1 (true), 0 (false)
NRCMODE	1 (true), 0 (false)
NRCLANG	usascii, frenchcanadian

---

<b>Keywords for Digital VT</b>	<b>Valid Setting(s)</b>
MODE	vt52, vt100, vt101, vt102, vt220, vt320, vt420
SCREENROW	24, 25, 36, 48
COLUMNS	80, 132
BITCONTROL	7, 8
DESTRUCTIVEBS	1 (true), 0 (false)
ANSWERBACK	string
GIEXTENSION	1 (true), 0 (false)
DISPLAYCONTROLSMODE	1 (true), 0 (false)
LOCALECHO	1 (true), 0 (false)
LINEWRAP	1 (true), 0 (false)
NEWLINE	1 (true), 0 (false)
NUMPADAPPLICATION	1 (true), 0 (false)
CURSORKEYAPP	1 (true), 0 (false)
LIGHTSCREEN	1 (true), 0 (false)

---

---

## EMULCONFIG, *continued*

---

Keywords for TVI 925/950	Valid Setting(s)
MODE	Full Duplex, Half Duplex, Block, Local
LINESPERPAGE	24, 48, 96
COLUMNS	80, 132
NUMBEROFPAGES	1, 2, 4
MODEL	TV925, TV950
ANSWERBACK	string
LIGHTSCREEN	1 (true), 0 (false)
DISPLAYCONTROLS	1 (true), 0 (false)
LOCALEEDITKEYS	1 (true), 0 (false)

Keywords for WYSE	Valid Setting(s)
MODE	wyse50, wyse60
ENDOFBLOCK	uscr, crlfetx
COMMMODE	full, block, half, halfblock
CLEARONWIDTHCHANGE	1 (true), 0 (false)
LOCALECHO	1 (true), 0 (false)
LINEWRAP	1 (true), 0 (false)
RECEIVECR	1 (true), 0 (false)
AUTOSCROLL	1 (true), 0 (false)
SOUNDS	1 (true), 0 (false)
MARGINBELL	1 (true), 0 (false)
ANSWERBACK	string



---

## EMULCONFIG, *continued*

---

If the AT&T 4425 Client Option is installed, these keywords are available:

Keywords for AT&T4425	Valid Setting(s)
RETURNKEYSENDS	CR, LF, CRLF
COLUMNS	80, 132
LOCALECHO	1 (true), 0 (false)
AUTOWRAP	1 (true), 0 (false)
ANSWERBACK	string
NEWLINEONLINEFEED	1 (true), 0 (false)
DISABLEAUXPRINTMODE	1 (true), 0 (false)

If the HP 700/94 Client Option is installed, these keywords are available:

Keywords for HP 700/94	Valid Setting(s)
BELL	1 (true), 0 (false)
INVERSEBACKGROUND	1 (true), 0 (false)
XMITFNCTN	1 (true), 0 (false)
INHEOLWRP	1 (true), 0 (false)
INHDC2	1 (true), 0 (false)
AUTOLF	1 (true), 0 (false)
TABSPACES	1 (true), 0 (false)
WARNINGBELL	1 (true), 0 (false)
LOCALECHO	1 (true), 0 (false)
SPOW	1 (true), 0 (false)
INHHNDSHK	1 (true), 0 (false)
ESCXFER	1 (true), 0 (false)
RETURNENTER	1 (true), 0 (false)
MODEL	HP700/92, HP700/94
COLUMNS	80, 132
STARTCOL	1 – 80
LINEPAGE	line, page
BLOCKMODE	1 (true), 0 (false)
NUMPADTAB	tab, return, enter
DECIMALTYPE (HP 700/94 only)	US, EUR
PRINT	all, fields
IMPDECDIGITS (HP 700/94 only)	0 – 9
TRANSMIT (HP 700/94 only)	all, modified
ATTRF1	N, L, T
ATTRF2	N, L, T
ATTRF3	N, L, T

## EMULCONFIG, *continued*

Keywords for HP 700/94	Valid Setting(s), <i>continued</i>
ATTRF4	N, L, T
ATTRF5	N, L, T
ATTRF6	N, L, T
ATTRF7	N, L, T
ATTRF8	N, L, T
LABELF1	string (0 – 16 characters)
LABELF2	string (0 – 16 characters)
LABELF3	string (0 – 16 characters)
LABELF4	string (0 – 16 characters)
LABELF5	string (0 – 16 characters)
LABELF6	string (0 – 16 characters)
LABELF7	string (0 – 16 characters)
LABELF8	string (0 – 16 characters)
STRINGF1	string
STRINGF2	string
STRINGF3	string
STRINGF4	string
STRINGF5	string
STRINGF6	string
STRINGF7	string
STRINGF8	string
AUTOLF (on-screen function key)	1 (true), 0 (false)
BLOCKMODE (on-screen function key)	1 (true), 0 (false)
DISPLAYFUNCTIONS (on-screen function key)	1 (true), 0 (false)
FORMATMODE* (on-screen function key)	1 (true), 0 (false)
KEYBOARDLOCK* (on-screen function key)	1 (true), 0 (false)
LINEMODIFY* (on-screen function key)	1 (true), 0 (false)
MEMORYLOCK* (on-screen function key)	1 (true), 0 (false)
MODIFYALL* (on-screen function key)	1 (true), 0 (false)
REMOTEMODE (on-screen function key)	1 (true), 0 (false)

\* when using these keywords with the GETEMULCONFIG function, the state of these on-screen keys can only be correctly returned when the session is connected; when the session is not connected these keywords will always return “FALSE”, even if set to “TRUE”.

---

## EMULCONFIG, *continued*

---

If the Tandem 6530 Client Option is installed, the following keywords and valid settings are available:

Keywords for Tandem 6530	Valid Setting(s)
EXEC_FUNCTION	1 (true), 0 (false)
RETURN_FUNCTION	1 (true), 0 (false)
BELL_COLUMN	0 (disables) 1-80
ALLOWHOSTCOLORS	1 (true), 0 (false)
EM3270_MODE	1 (true), 0 (false)
LOCAL_TRANSMIT_COLUMN	1-80
BLOCK_TYPE_AHEAD	1 (true), 0 (false)
CI_ERROR_RESPONSE	no_response, error_msg_only, del_symbol_ only, del_symbol_and_error_msg
NATIONAL_LANGUAGE_SUPPORT_8_BIT	1 (true), 0 (false)
LANGUAGE_8	Westeurp_8 (also known as Latin_1_8), Cyril- lic_8, or Greek_8
NATIONAL_LANGUAGE_SUPPORT_7_BIT	1 (true), 0 (false)
LANGUAGE_7	USASCII, French_7, German_7, Spanish_7, UK_7, Swedish_7, Danish_7, Norwegian_7, Belgian_7, Portuguese_7
AUX1 or AUX2	none, printer, file, device
AUX1_DEVICE or AUX2DEVICE	File name or device name (up to 256 charac- ters)

---

## EMULCONFIG, *continued*

If the IBM TN3270 Client Option is installed, the following keywords and valid settings are available:

Keywords for TN3270	Valid Setting(s)
FS3270_TermType	IBM3278, IBM3279
FS3270_TermModel	2, 3, 4, 5
FS3270_ExtAttrib	1 (true), 0 (false)
FS3270_OEMReply	1 (true), 0 (false)
FS3270_AltSize	1 (true), 0 (false)
FS3270_Sound	1 (true), 0 (false)
FS3270_Notify	1 (true), 0 (false)
FS3270_TypeAhead	1 (true), 0 (false)
FS3270_LongName	String, A sequence of up to eight characters, which describe the session and its connection.
FS3270_LUName	A sequence of up to eight characters. The name must be unique among other logical unit names, is case sensitive, and can consist of numbers, uppercase letters, and special characters (such as the following symbols: percent %, dollars \$, number #, and at sign @). This name is not required, unless DCS is connecting via the Microsoft SNA Server connector.
FS3270_CharacterSet	<i>This string is not case sensitive, but the characters should match the names in the list box.</i>  Austrian/German, Austrian/German CECP, Belgian, Belgian CECP, Canadian Bilingual, Canadian Bilingual CECP, Danish, Danish CECP, English- UK, English- US, English- US (Old), English- US C370 CECP, English- US C370 CECP V2, English- US CECP, Finnish, Finnish CECP, French, French CECP, Italian, Italian CECP, Netherlands, Netherlands CECP, Norwegian, Norwegian CECP, Portuguese, Portuguese CECP, Spanish CECP, Spanish Speaking, Spanish Speaking CECP, Swedish, Swedish CECP, Swiss French, Swiss French CECP, Swiss German, Swiss German CECP
FS3270_Uppercase	1 (true), 0 (false)
FS3270_AutoSkip	1 (true), 0 (false)
FS3270_ConvertNulls	1 (true), 0 (false)
FS3270_DisplayNulls	1 (true), 0 (false)
FS3270_RespectNumeric	1 (true), 0 (false)
FS3270_GraphicalOIA	1 (true), 0 (false)
FS3270_TLS	yes (enabled)

---

## EMULCONFIG, *continued*

---

If the IBM TN5250 Client Option is installed, the following keywords and valid settings are available:

Keywords for TN5250	Valid Setting(s)
FS5250_TermModel	2, 5
FS5250_Sound	1 (true), 0 (false)
FS5250_Notify	1 (true), 0 (false)
FS5250_TypeAhead	1 (true), 0 (false)
FS5250_LongName	String, A sequence of up to eight characters, which describe the session and its connection.
FS5250_CharacterSet	<i>This string is not case sensitive, but the characters should match the names in the list box.</i>  Austrian/German, Austrian/German CECP, Belgian, Belgian CECP, Canadian Bilingual, Canadian Bilingual CECP, Danish, Danish CECP, English- UK, English- US, English- US (Old), English- US C370 CECP, English- US C370 CECP V2, English- US CECP, Finnish, Finnish CECP, French, French CECP, Italian, Italian CECP, Netherlands, Netherlands CECP, Norwegian, Norwegian CECP, Portuguese, Portuguese CECP, Spanish CECP, Spanish Speaking, Spanish Speaking CECP, Swedish, Swedish CECP, Swiss French, Swiss French CECP, Swiss German, Swiss German CECP
FS5250_UpperCase	1 (true), 0 (false)
FS5250_DisplayNulls	1 (true), 0 (false)
FS5250_GraphicalOIA	1 (true), 0 (false)

### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

The **WINDOW** clause applies the emulation configuration settings to a particular session window.

---

## EMULCONFIG, *continued*

---

### Comments

The keywords correspond to parameters available on the **Emulations** tab of the **Session Properties** dialog. This command, therefore, allows you to set these parameters through the DCS script language rather than using the dialog box.

If the **WinHandle** is not specified, the emulation configuration settings are applied to the active session window.

The **ERROR** function returns **TRUE** if the **WinHandle** or **String** keyword is invalid.

### Example

In this example:

```
EMULCONFIG "Columns=80"
```

the command sets the emulator screen column width to 80.

---

# END

---

## END

The END command specifies the end of a command block.

### Arguments

The END command takes no arguments.

### Comments

The commands in the command block must be preceded by a BEGIN command.



Also see: [BEGIN command](#)

### Example

This script example:

```
.  
.   
.   
RECORD READ 0  
WHILE NOT EOF (  
BEGIN  
DISPLAY @R0 | ``^M``  
RECORD READ 0  
END  
.   
.   
.
```

assumes a table has been defined with the TABLE DEFINE command earlier in the script. In this script segment shown, the BEGIN and END commands are used to define a block of commands that execute repeatedly as long as the WHILE command is TRUE.

---

# EXECUTE

---

## EXECUTE Target

The EXECUTE command restarts script execution with a script or script routine (as if you had stopped the script and selected **Execute** on the **Script** menu).

### Arguments

#### Target

The **Target** argument is specified by either a near target (usually a routine in the current script) or a far target (usually another script or a routine in another script).

### Comments

The execution of an EXECUTE command clears any existing resources created during script execution (variables, tables, etc.).

### Example

In this example:

```
IF RADIOGROUP (1) = 3
EXECUTE "values*add_values"
```

if the third radio button in RADIOGROUP 1 (one) is selected, execution branches to the far target `values*add_values`. This causes execution to resume in the script `values` at the label `add_values`.



---

# FILE CLOSE

---

## FILE CLOSE WINDOW WinHandle

The FILE CLOSE command terminates the text transfer process initiated by a LOGTOFILE command.



Note: This command does not apply to IBM TN3270 emulations.

### Arguments

#### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular child window.

### Example

In this example:

```
DIAL '5552222'  
PERFORM login_sequence  
LOGTOFILE 'ONLINE.TXT'  
PERFORM get_data  
FILE CLOSE
```

DCS logs onto the dial-up service and prepares to capture incoming data. When all the data has been received, the FILE CLOSE command saves the data in the file ONLINE.TXT and then closes the file.

---

## FILE COMPRESS

---

### FILE COMPRESS FileName

The FILE COMPRESS command converts an eight-data-bits-per-byte file into a seven-data-bits-per-byte file.

#### Arguments

##### FileName

The FileName argument is a string specifying the file to be compressed. The FileName argument must specify a valid file name for your system.

#### Comments

The line length is limited to a maximum of 130 characters, and all control characters (except carriage returns) are converted to ASCII characters. The FILE COMPRESS command converts a binary file to an ASCII text file, allowing it to be transferred by a protocol that utilizes only seven data bits.

The ERROR function returns TRUE if DCS cannot compress the file. DCS will not compress the file if it:

- ▼ Cannot open the file (for example, if the file does not exist)
- ▼ Cannot open a temporary file (for example, if there is not enough free space on your hard drive)
- ▼ Cannot replace the contents of the original file (for example, if the file is read-only),
- ▼ If either a read or write error occurs during the compression.

#### Example

See the FILE ENCRYPT and the FILE DECOMPRESS commands.

---

## FILE COPY

---

### FILE COPY Source TO Destination APPEND

The FILE COPY command copies the contents of the source file to the destination file, creating the destination file if it does not exist.

#### Arguments

##### Source

The Source argument is a string specifying the file containing the data to be copied. The Source argument must specify a valid file name for your system.

##### TO

The TO clause allows the Destination argument to be specified for the Source file. The clause is composed of the keyword TO and the Destination argument.

##### Destination

The Destination argument is a string specifying the file to which the data is to be copied. The Destination argument must specify a valid file name for your system.

##### APPEND

The optional APPEND keyword causes the data to be appended to the existing contents of the destination file. If APPEND is not included, DCS clears the content of the existing file before copying the contents of the source file.

#### Comments:

The ERROR function returns TRUE if the source file does not exist.

#### Example

In this example:

```
*moveFiles ($SourceFile, $DestPath, #Erase)
ARGUMENTS ($NextFile)
$NextFile = ROUTE ($SourceFile)
WHILE NOT ERROR ()
BEGIN
FILE COPY $NextFile TO $DestPath
IF #Erase FILE
FILE DELETE $DestPath
$NextFile = Next ()
END
```

the WHILE NOT ERROR subroutine takes a source file specification (\$SourceFile), a destination path (\$DestPath), and a Boolean flag (#Erase), and copies the source files to the specified directory (with an option to delete the source file).

---

# FILE CREATENAME

---

**FILE CREATENAME** **FileName** **TYPE** **Type** **DEFAULT** **String** **TITLE** **TitleText** **NOCREATE**

The FILE CREATENAME command displays the **Save As** dialog listing the available files, and prompts for the name of the file to create. The file name is stored in a string variable.

## Arguments

### FileName

The **FileName** argument is a string variable in which DCS stores the file name entered in the **Save As** dialog box displayed for this command. If you enter a file name which did not previously exist, DCS stores the new file name in the **FileName** argument and create an empty file in the current working directory.

If you enter a file name which exists in the current directory, DCS asks you whether it should replace the contents of the file located in the directory. If you click the **Yes** button, DCS deletes the previous contents of the file and place the file name in the **FileName** variable. If you click the **No** button, DCS puts a null character into the **FileName** variable.

However, if you have included the optional **NOCREATE** keyword with this command, DCS does not ask you whether it should delete the contents of previously existing files. With the **NOCREATE** keyword, DCS opens the file to allow you to append characters to or to delete characters from the file using the **TABLE** and **RECORD** commands.

### TYPE Type

The optional **TYPE** clause lists only files of a particular type in the **Save As** dialog displayed for this command. If you do not include this clause, DCS lists files of all types. The **TYPE** argument is a string that specifies the type of files to list.

The **TYPE** argument must have the following format:

“FileDescription1#FilterSpec1#FileDescription2#FilterSpec2#...”

Each element of this string must end with a number sign (#). **FileDescription $n$**  and **FilterSpec $n$**  are the elements of the string, and a **FilterSpec $n$**  element and its numeral sign must follow each **FileDescription $n$**  element and its numeral sign. The **FileDescription $n$**  and the **FilterSpec $n$**  elements form a unit, such that **FileDescription1** is associated with **FilterSpec1**, and that **FileDescription2** is associated with **FilterSpec2**, and so on.

The text contained in a **FileDescription $n$**  element appears in a **List of Files of Type** drop-down list box. A **FilterSpec $n$**  element is composed of characters DCS uses as a criterion to determine which file names in a directory to display in the **File Name** list box. For example, when you select the text from **FileDescription1** in the **List of Files of Type** drop-down list box, DCS uses the characters from **FilterSpec1** to determine which files to show in the **File Name** list box, and similarly with **FileDescription2**, DCS uses the characters from **FilterSpec2**, and so on.

The **FilterSpec1** element will become the default text for the **File Name** edit text box in the file open dialog box displayed for this command, if you have not included the **DEFAULT** clause in the command.

---

## FILE CREATENAME, *continued*

---

The format of the characters in a `FilterSpecn` element must be valid for your system. Wildcard characters, the asterisk and the question mark, are acceptable. For example, `FilterSpecn` might be one of the following:

<code>*.D*</code>	Displays all of the file names whose extension start with D.
<code>M*.TXT</code>	Displays all text file names which start with M.
<code>L?????.DC*</code>	Displays all file names which begin with L and have five characters after the L and whose extensions start with DC.

### DEFAULT String

The optional **DEFAULT** clause directs DCS to display the contents of the **String** argument in the **File Name** edit text box when DCS initially displays the **Save As** dialog for this command. If you have not included the **DEFAULT** clause, DCS will display the `FileSpec1` element from the **TYPE** clause (if you have included the **TYPE** clause in the command) or will leave the **File Name** edit text box empty initially. The **String** argument must be a valid file name for your system. Do not place wildcard characters in the **String** argument.

### TITLE TitleText

The optional **TITLE** clause directs DCS to display the contents of the **TitleText** argument in the title bar of the **Open** dialog displayed for this command. The **TitleText** argument is a string specifying the text for the title bar.

### NOCREATE

If you have include the optional **NOCREATE** keyword with this command, DCS does not ask you whether it should delete the contents of previously existing files. With the **NOCREATE** keyword, DCS opens the file to allow you to append characters to or to delete characters from the file using the **TABLE** and **RECORD** commands.

### Comments

The **ERROR** function returns **TRUE** if you decide not to create a file. For example, if you select the **Cancel** button in the **Open** dialog DCS displays, or if you select a file name that exists in a directory and you choose not to overwrite the file, the **ERROR** function returns **TRUE**.

A file name and its path may have a maximum of 256 characters.

---

## FILE CREATENAME, *continued*

---

### Example

This command:

```
FILE CREATENAME $FileName TYPE "Data Files (*.DAT)#\  
*.DAT#Mail Files (TXT)#M*.TXT#Scripts (Text & \  
Task)#*.DC*#" TITLE "DCS's Files"
```

displays the Save As dialog . By default, DCS selects Data Files (\*.DAT) in the **List Files of Type** drop-down list box, displays the text \*.DAT in the **File Name** edit text box, and displays the file names with a DAT extension contained in the current working directory. The title of the displayed **Open** dialog box is DCS Files. When you place a file name in the **File Name** edit text box and then select the **OK** button, DCS creates an empty file and gives it a name based on the characters in the edit text box, places the new file in the directory you have selected in the **Directories** list box, and then places the name of the file in the \$FileName variable.

This command:

```
FILE CREATENAME $file TYPE "*.DCP"
```

opens a **Save As** dialog which displays a list box containing all files with a DCP file extension in the current directory. When a file is selected, or a new file name has been entered, the file name is stored in the string variable \$file.

---

# FILE DECOMPRESS

---

## FILE DECOMPRESS FileName

The FILE DECOMPRESS command restores a file compressed with the FILE COMPRESS command to its original state.

### Arguments

#### FileName

The FileName argument is a string specifying the file to be decompressed. The FileName argument must specify a valid file name for your system.

### Comments

The ERROR function returns TRUE if DCS cannot decompress the file. DCS will not decompress the file if it:

- ▼ Cannot open the file (for example, if the file does not exist)
- ▼ Cannot open a temporary file (for example, if there is not enough free space on your hard drive)
- ▼ Cannot replace the contents of the original file (for example, if the file is read-only)
- ▼ If a read or write error occurs during the decompression.

### Example

See the FILE ENCRYPT command.

---

# FILE DECRYPT

---

## FILE DECRYPT FileName Key

The FILE DECRYPT command modifies the contents of the specified encrypted file to be readable.

### Arguments

#### FileName

The FileName argument is a string specifying the file to be decrypted. The FileName argument must specify a valid file name for your system.

#### Key

The Key argument is a string used for file decryption. It must be the same string specified as the Key argument in the FILE ENCRYPT command.

### Comments

The ERROR function returns TRUE if :

- ▼ The file is not encrypted
- ▼ The file does not exist
- ▼ The password is incorrect
- ▼ ENCRYPT.DLL cannot load (for example, if your computer has little free memory)
- ▼ DCS cannot create a temporary file
- ▼ A read or write error occurs during the decryption
- ▼ DCS cannot overwrite the original file (for example, if the file is read-only).

### Example

See the FILE ENCRYPT command.



---

## FILE DELETE

---

### FILE DELETE FileName

The FILE DELETE command deletes the specified file or directory.

#### Arguments

##### FileName

The FileName argument is a string specifying the file (or directory) to be deleted. The FileName argument must specify a valid file name for your system.

#### Comments

The ERROR function returns TRUE if the file does not exist.

#### Example

See the FILE COPY command.

---

# FILE ENCRYPT

---

## FILE ENCRYPT FileName Key

The FILE ENCRYPT command modifies the contents of the specified file, making it appear to be unreadable unless decrypted using the FILE DECRYPT command.

### Arguments

#### FileName

The FileName argument is a string specifying the file to be encrypted. The FileName argument must specify a valid file name for your system.

#### Key

The Key argument is a string, which must be used as the Password argument of a FILE DECRYPT command in order to decrypt the file.

### Comments

The ERROR function returns TRUE if:

- ▼ The file does not exist
- ▼ ENCRYPT.DLL cannot load (for example, if your computer has little free memory)
- ▼ DCS cannot create a temporary file
- ▼ A read or write error occurs during the encryption
- ▼ DCS cannot overwrite the original file (for example, if the file is read-only).

### Example

In this example:

```
FILE ENCRYPT "MYFILE.DCP" "secretcode"  
FILE COMPRESS "MYFILE.DCP"  
FILE SEND BINARY "MYFILE.DCP"  
. . .  
FILE RECEIVE BINARY "MYFILE.DCP"  
FILE DECOMPRESS "MYFILE.DCP"  
FILE DECRYPT "MYFILE.DCP" "secretcode"
```

a script file is encrypted to encode its contents and compressed so it can be sent using a text transfer. Later, at the FILE RECEIVE command, the host returns the file. For DCS to use the script file, it must first be decompressed and then decrypted.

---

## FILE OPENNAME

---

### FILE OPENNAME FileName TYPE Type DEFAULT String TITLE TitleText

The FILE OPENNAME command displays a file open dialog box listing the available files and prompts you for the name of a file to open. The command then stores the file name in a string variable.

#### Arguments

##### FileName

The **FileName** argument is a string variable that stores the file name entered in the **Open** dialog displayed for this command.

##### TYPE Type

The optional **TYPE** clause lists only files of a particular type in the **Open** dialog displayed for this command. If you do not include this clause, DCS will list files of all types. The **Type** argument is a string specifying the type of files for DCS to list.

The **Type** argument must have the following format:

“FileDescription1#FilterSpec1#FileDescription2#FilterSpec2#...”

Each element of this string must end with a number sign (#). **FileDescription $n$**  and **FilterSpec $n$**  are the elements of the string, and a **FilterSpec $n$**  element and its numeral sign must follow each **FileDescription $n$**  element and its numeral sign. The **FileDescription $n$**  and the **FilterSpec $n$**  elements form a unit, such that **FileDescription1** is associated with **FilterSpec1**, and that **FileDescription2** is associated with **FilterSpec2**, and so on.

The text contained in a **FileDescription $n$**  element will appear in a **List of Files of Type** drop-down list box. A **FilterSpec $n$**  element is composed of characters DCS uses as a criterion to determine which file names in a directory to display in the **File Name** list box. For example, when you select the text from **FileDescription1** in the **List of Files of Type** drop-down list box, DCS uses the characters from **FilterSpec1** to determine which files to show in the **File Name** list box, and similarly with **FileDescription2**, DCS uses the characters from **FilterSpec2**, and so on.

The **FilterSpec1** element will become the default text for the **File Name** edit text box in the file open dialog box displayed for this command, if you have not included the **DEFAULT** clause in the command.

The format of the characters in a **FilterSpec $n$**  element must be valid for your system. Wildcard characters, the asterisk and the question mark, are acceptable. For example, **FilterSpec $n$**  might be one of the following:

- |                   |   |
|-------------------|---|
| <b>*.D*</b>       | Displays all of the file names whose extension start with D.  |
| <b>M*.TXT</b>     | Displays all text file names which start with M.  |
| <b>L?????.DC*</b> | Displays all file names which begin with L and have five characters after the L and whose extensions start with DC. |

---

## FILE OPENNAME, *continued*

---

### DEFAULT String

The optional **DEFAULT** clause displays the contents of the **String** argument in the **File Name** edit text box when the File Open dialog displays for this command. If you have not included the **DEFAULT** clause, DCS will display the **FileSpec1** element from the **TYPE** clause (if you have included the **TYPE** clause in the command) or will leave the edit text box empty initially. If you want the contents of the **String** argument to appear in the **File Name** edit text box, the **String** argument must be a valid file name and should not contain any wildcard characters.

If the **String** argument contains a valid path, DCS uses that path as the default directory from which to display files.

### TITLE TitleText

The optional **TITLE** clause directs DCS to display the contents of the **TitleText** argument in the title bar of the **Open** dialog displayed for this command. The **TitleText** argument is a string specifying the text for the title bar.

### Comments

The **ERROR** function returns **TRUE** when you cancel the **Open** dialog DCS displays for this command.

### Example

This command:

```
FILE OPENNAME $FileName TYPE "Data Files (*.*)# *.DAT#" \
Default "*.DAT" TITLE "Data Files"
```

displays an **Open** dialog. By default, DCS selects Data Files (\*.DAT) in the **List Files of Type** drop-down list box, displays the text \*.DAT in the **File Name** edit text box, and displays in the **File Name** list box the files with a DAT extension contained in the current working directory. The title of the **Open** dialog that DCS displays is "Data Files". When you enter a file name in the **File Name** edit text box and then select **OK**, DCS places the name of the file in the **\$FileName** variable.

---

## FILE PAUSE

---

### FILE PAUSE

The FILE PAUSE command temporarily suspends the text transfer initiated by the active LOGTOFILE command, until a FILE RESUME command is executed, or **New** on the **File** menu is selected.



Note: This command does not apply to IBM TN3270 emulations.

### Arguments

The FILE PAUSE command takes no arguments.

### Example

In this example:

```
DIAL
PERFORM login
;connects and performs login to remote system
LOGTOFILE "VACATION.TXT"
;captures data from host to the file VACATION.TXT
PERFORM OAG
;perform commands that retrieve data from OAG System
FILE PAUSE
;pause the text file transfer
PERFORM checkmail
;performs commands that retrieve new mail from host system
FILE RESUME
;resumes the test file transfer
PERFORM hotel
;performs commands that retrieve hotel information from
the hosts system
FILE CLOSE "VACATION.TXT"
;terminates the file transfer
```

DCS logs onto an online system and begins saving the session in the file VACATION.TXT. After performing the OAG subroutine, DCS stops recording the session and executes the checkmail subroutine. When the FILE RESUME command executes, DCS resumes recording the session until it saves and closes VACATION.TXT.

---

## FILE RECEIVE BINARY

---

**FILE RECEIVE REMOTE BINARY FileName AS Dest ASCII BINARY CRLF NOCRLF NEITHER APPEND WINDOW WinHandle**

The FILE RECEIVE BINARY command prepares DCS to receive the specified file using the binary file transfer protocol specified in the Session Properties of the active session.

### Arguments

#### REMOTE

The optional **REMOTE** keyword causes the FILE RECEIVE BINARY command to issue the Kermit GET command to a Kermit server on a non-3270 remote host. The Kermit binary transfer protocol must be specified for the session before executing the FILE RECEIVE BINARY command with the **REMOTE** keyword.

For the Kermit protocol, if the remote system is in server mode then you must use the keyword **REMOTE** .

#### BINARY FileName

The FileName argument is a string specifying the file in which to store the incoming file. The FileName argument must specify a valid file name for your system.

The FileName argument might be in one of the following formats:

CMS: HostFileName HostFileType HostFileMode

TSO: HostFileName HostMemberName Password

The FileName argument must be the same as the file name on the remote system for the Kermit protocol only. You may rename the file on your PC using the **AS Dest** clause.

The Kermit, YModem, and ZModem transfer protocols support transfers of multiple files, or batch transfer. To receive multiple files, you should set the FileName argument to an empty string (""), not a wildcard string (\*.\*) .

#### AS Dest

The **AS** clause is an optional argument for transfers involving non-3270 terminal emulations, but is required for transfers involving 3270 emulations. This clause allows you to save the file under a name different from the name on the remote system. The **Dest** argument is a string specifying the name under which the received file is stored. The **Dest** argument must specify a valid file name on your system.

When DCS emulates a non-3270 terminal, the command only recognizes this clause when you have also configured the active session window to use the Kermit transfer protocol as its binary transfer protocol. When the Kermit protocol is not configured for a session window, or when you do not include the **AS** clause, DCS saves the file under the same name that the file had on the remote system.

---

## FILE RECEIVE BINARY, *continued*

---

### ASCII

The optional **ASCII** keyword is for IND\$File transfers only. This keyword specifies an EBCDIC to ASCII translation for text files before a text file is stored on the local computer. If the option is not included, the file is not translated. Do not include this option if the file is already in ASCII format.

### CRLF

The optional **CRLF** keyword is for IND\$File transfers only. This keyword directs DCS to replace all end of record marks with a carriage return character followed by a line feed character. If you do not include this keyword, end of record marks are removed and not replaced.

### APPEND

The optional **APPEND** keyword is for IND\$File transfers only. This keyword causes DCS to append the incoming file to the existing contents of the specified file and to override the record length and record format specifications. If it is not included, DCS clears the existing contents of the file before the file is received. This option is used only when transferring files to a TSO or VM/CMS command processor.

### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular child window.

The **WINDOW** clause directs DCS to receive a file using the file transfer configuration of a particular session.

### Comments

If the remote system does not begin sending the specified file within 30 seconds, DCS aborts the transfer and displays a dialog box indicating it aborted the transfer.

For systems using the Kermit protocol, the **FILE RECEIVE REMOTE BINARY** command is equivalent to a **Kermit GET** command.

The **ERROR** function returns **TRUE** if you abort the transfer, or if DCS aborts the transfer, due to too many transmission errors. When there is an error, DCS places text describing the error in the **Result** system variable.

If the **WINDOW** clause is not included in the command, the file is received through the current session.

---

## FILE RECEIVE BINARY, *continued*

---

### Example

In this example:

```
SET BINARYTRANSFERS ZMODEM
FILE RECEIVE BINARY "MEMO.TXT"
```

DCS prepares to receive all incoming files via ZModem, and then download the file MEMO . TXT.

In this example:

```
SET BINARYTRANSFER KERMIT
SEND "KERMIT"
WAIT STRING "KERMIT-32>"
SEND "SERVER"
WAIT STRING "COMMAND"
FILE RECEIVE REMOTE BINARY "LOG.COM" AS "LOG.TXT"
KERMIT FINISH
SEND "EXIT"
```

DCS instructs the remote system to enter Kermit server mode. When the remote system is in Kermit server mode, your system can send commands to perform operations, such as viewing or transferring files, on the remote system. In this example, DCS instructs the remote system to send the file LOG . COM. The AS LOG . TXT clause directs DCS to rename the received file on your PC. The KERMIT FINISH command instructs the remote system to leave server mode, and the string EXIT causes the remote system to exit the Kermit application.

```
FILE RECEIVE BINARY "File" "data" "A1" AS "data.TXT"
```

In this example, DCS prepares to receive a file from a remote system with the CMS environment. DCS receives the specified host file and stores it in DATA . TXT.

```
File Receive Binary "datfile" "" "pass" AS "datfile.tx"
```

In this example, DCS prepares to receive a file from a remote system running the TSO environment. The command directs DCS receives the host file and stores it in datfile . tx. Since the host file is not from a partitioned data set (the host file has no member name) a null string ("") stands for the HostMemberName argument.



---

# FILE RENAME

---

## FILE RENAME Source TO Destination

The FILE RENAME command changes the name of the file specified by `Source` to the name specified by `Destination`.

### Arguments

#### Source

The `Source` argument is a string specifying the file to be renamed. The `Source` argument must specify a valid file name for your system.

#### Destination

The `Destination` argument is a string specifying a new file name. The `Destination` argument must specify a valid file name for your system.

### Comments

The `ERROR` function returns `TRUE` if the source file does not exist or if the destination file already exists.

### Example

In this example:

```
IF EXISTS ("OLDDATA")
  FILE DELETE "OLDDATA"
FILE RENAME "MYDATA" TO "OLDDATA"
FILE RECEIVE BINARY "MYDATA"
```

the current contents of the file `MYDATA` is protected by renaming the file to `OLDDATA` before receiving a binary file of the same name.


---

## FILE RESUME

---

### FILE RESUME WINDOW WinHandle

The FILE RESUME command restarts the text transfer suspended by a FILE PAUSE command. It affects only the active LOGTOFILE command.

 Note: This command does not apply to IBM TN3270 emulations.

### Arguments

#### WINDOW WinHandle

The optional **WINDOW** clause is composed of both the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular child window.

### Comments

If the active LOGTOFILE command is performed in line count mode, script execution pauses until DCS transfers the specified number of lines or until the transfer is completed.

### Example

See the FILE PAUSE command.

---

## FILE SEND BINARY

---

### **FILE SEND REMOTE BINARY FileName AS Hostname ASCII BINARY CRLF NOCRLF APPEND WINDOW WinHandle**

The FILE SEND BINARY command directs DCS to send the specified file using the binary transfer protocol specified by the active settings.

#### **Arguments**

##### **REMOTE**

The optional **REMOTE** keyword causes the FILE SEND BINARY command to issue the KERMIT PUT command to a Kermit server on a remote host. The Kermit binary transfer protocol must be specified for the session window before executing the FILE SEND BINARY command with the **REMOTE** keyword. For the Kermit file transfer protocol, you must use the **REMOTE** keyword if the remote system is in server mode.

##### **BINARY FileName**

The **FileName** argument is a string specifying the file to send. The **FileName** argument must specify a valid file name for your system.

##### **AS Hostname**

Only use the **AS** clause with the Kermit protocol. The **AS** clause is an optional argument for transfers involving non-3270 terminal emulations, but is required for transfers involving 3270 emulations. This clause allows you to save the file on the remote system under a name different from the current file name. The **Hostname** argument is a string specifying the name to give the file on the remote system. The **Hostname** argument must specify a valid file name for the remote system.

When DCS emulates a non-3270 terminal, the command only recognizes this clause when you have also configured the active session window to use Kermit as its file transfer protocol. When the Kermit protocol is not configured for a session window, or when you do not include the **AS** clause, DCS saves the file using its current file name.

The **Hostname** argument might be in one of the following formats:

CMS: HostFileName HostFileType HostFileMode

TSO: HostFileName HostMemberName Password

If you do not include the **AS** clause, the remote system will save the file under the same name it had on your PC.

##### **ASCII**

This option is for IND\$File transfers only. The optional **ASCII** keyword specifies an EBCDIC to ASCII translation. If it is not included, the file is not translated.

##### **CRLF**

This option is for IND\$File transfers only. The optional **CRLF** keyword directs DCS to replace each end of line delimiter (a carriage return character followed by a line feed character) with an end of record mark.

---

## FILE SEND BINARY, *continued*

---

### APPEND

This option is for IND\$File transfers only. The optional **APPEND** keyword causes DCS to append the outgoing file to the existing contents of the specified file and to override the record length and record format specifications. If it is not included, DCS clears the existing contents of the file before receiving the file. This option is included when transferring files to a remote system with a TSO or VM/CMS command processor.

### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

The **WINDOW** clause directs DCS to send a file as configured in a specific session.

### Comments

If the remote system does not respond to your transmission within 30 seconds, DCS will display a dialog box indicating that it aborted the file transfer.

For systems using the Kermit protocol, the **FILE SEND REMOTE BINARY** command is equivalent to a **KERMIT PUT** command.

The **ERROR** function returns **TRUE** if you abort the transfer, or if DCS aborts the transfer due to too many transmission errors. When there is an error, DCS will place text describing the error in the **Result** system variable.

If the **WINDOW** clause is not included in the command, the file is sent through the current session.

### Example

In this example:

```
FILE SEND BINARY "TEST.FIL" AS "TEST" "FILE" "A1"
```

DCS sends a file to a 3270 host running a CMS environment. The command directs DCS to send the file **TEST.FIL** to the host and to store it in the file **TEST FILE A1**.

In this example:

```
SET BINARYTRANSFERS YMODEM  
FILE SEND BINARY "C:\DCSERIES\*.TXT"
```

DCS sends several files via YModem in batch mode. All files in the **DCSERIES** directory on drive **C:** that have a **TXT** file extension will be sent.

---

## FILE SEND BINARY, *continued*

---

In this example:

```
SET BINARYTRANSFERS XMODEM
SET BAUDRATE 14400
CONNECT
PERFORM login
FILE SEND BINARY "DCDATA.TXT"
```

DCS sends the file DCDATA.TXT after the appropriate binary transfer settings have been selected.

In this example:

```
SET BINARYTRANSFERS KERMIT
SEND "KERMIT"
WAIT STRING "KERMIT-32>"
SEND "SERVER"
WAIT STRING "COMMAND"
FILE SEND REMOTE BINARY "LOGIN.COM" AS "LOGIN.GDT"
KERMIT FINISH
SEND "EXIT"
```

DCS instructs the remote system to enter Kermit server mode. When the remote system is in Kermit server mode, your system can send commands to perform operations, such as viewing or transferring files, on the remote system. DCS sends the file LOGIN.TXT to the remote system, and instructs the remote system to name the file to LOGIN.GDT. The KERMIT FINISH command instructs the remote system to leave Kermit server mode, and the EXIT string causes the remote system to exit the Kermit application.

In this example:

```
FILE SEND BINARY "datafile.TXT" AS "datafile" "" "pass"
```

DCS sends a file to a remote system running the TSO environment. The command directs DCS to send the file DATAFILE.TXT to the remote system, stores it in the file named DATAFILE, and uses the password pass. The null string ("") is used for the HostMember-Name argument since the host file is not from a partitioned data set (it has no member name).

---

# FKEYS

---

## FKEYS Display

The FKEYS command controls whether the session toolbar is displayed in a session window.

### Arguments

#### Display

The Display argument is specified by one of the following keywords: **SHOW** or **HIDE**. The **SHOW** keyword directs DCS to display the session toolbar. The **HIDE** keyword directs DCS to remove the function key display.

### Comments

You may still access hidden session toolbar buttons by using the proper accelerator key sequence.

### Example

In this example:

```
LOAD "LOGON.SES"  
FKEYS SHOW  
PERFORM OnlineSession  
FKEYS HIDE
```

DCS loads the session file `LOGON.SES`. The session toolbar buttons are displayed for use during the online session that follows, and then hidden when the session is completed.

---

# GENERALCONFIG

---

## GENERALCONFIG String WINDOW WinHandle

The GENERALCONFIG command sets the value of a parameter used to configure general session options.

### Arguments

#### String

The String argument represents a single “keyword” followed by the assignment operator (=) and a valid setting. Together, the keyword and the setting are used to configure the general properties of the active session.

Keywords for General tab	Valid Setting(s)
AutoConnect	1 (true), 0 (false)
SaveWindowPosition	1 (true), 0 (false)
StartupScript	string of valid path of a compiled script file
DisplayDisconnectConfirmation	1 (true), 0 (false)
DisplaySaveConfirmation	1 (true), 0 (false)
DisplayConnectInformation	1 (true), 0 (false)
DisplayErrorInformation	1 (true), 0 (false)
Sound	1 (true), 0 (false)

### Comments

The keywords correspond to parameters available on the **General** tab of the **Session Properties** dialog. This command, therefore, allows you to set these parameters through the DCS script language rather than using the dialog box.

The ERROR function returns TRUE if the **WinHandle** or **String** keyword is invalid.

Changes made via scripting to these options apply only to the active or specified session and are not saved unless the session is saved.



**Note:** Because scripting tasks receive a lower processing priority than other application activity, a session which is already configured to autoconnect may do so upon opening before it receives the command from a script to disable the autoconnect feature.



**Also see:** GETGENERALCONFIG function

---

## GENERALCONFIG, *continued*

---

### Example

In this example:

```
LOAD
GENERALCONFIG "AutoConnect=false"
GENERALCONFIG "StartupScript=p:\public\scripts\
vaxlogin.dct"
GENERALCONFIG "DisplaySaveConfirmation=false"
SAVE "vax1.ses"
```

a new session is loaded and configured with the autoconnect option turned off. A login script (which can be used to specify the host system and initiate the connection) is specified to start once the session opens. Finally, the save confirmation dialog is disabled and the session is saved.



---

# GOTO

---

## GOTO Target

The GOTO command causes script execution to branch to the specified target.

### Arguments

#### Target

The Target argument can be specified by a near target (usually a routine in the local script) or a far target (another script, or a routine in another script).

### Example

In this example:

```
SET %listbox LISTBOX ()
IF %listbox < 0
GOTO error_display
ELSE
BEGIN
RECORD READ 0 AT %listbox
DIAL @R0.2 RETRY 1 DELAY 45
END
.
.
.
*error_display
.
.
.
```

if no item is selected in the list box, the GOTO command branches to an error handling routine.

---

# HANGUP

---

## HANGUP WINDOW WinHandle

The HANGUP command directs the modem to disconnect the telephone line.



Note: This command does not apply to IBM TN3270 emulations.

### Arguments

#### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

The **WINDOW** clause directs DCS to disconnect the phone line for a specific session.

### Comments

If the **WINDOW** clause is not included in the command, DCS will disconnect the phone line of the active session window.

### Example

In this example:

```
IF CONNECT ()  
HANGUP  
DIAL '5551234'
```

if DCS is currently connected, these commands direct DCS to disconnect the line before attempting to connect using another phone number.

---

# IF

---

## IF Boolean Command ELSE Command

The IF command evaluates the specified Boolean argument, executing the command list if the Boolean argument is TRUE.

### Arguments

#### Boolean

The Boolean argument specifies the Boolean to be evaluated.

#### Command

The Command argument specifies a logical command (either a single command or a command block).

#### ELSE Command

The optional ELSE clause executes the commands in the ELSE command list if the Boolean argument is FALSE. If the ELSE clause appears on the same line as the IF command, it must be preceded by a comma. The Command argument specifies a logical command (either a single command or a command block).

### Example

In this example:

```
RECORD READ 0 at 0
WHILE (TRUE)
BEGIN
IF EOF ( )
BEGIN
LEAVE
END
RECORD READ 0
END
DISPLAY "PROCESS COMPLETE^M"
CANCEL
```

if the end of the file is reached, DCS displays the specified string in the window and the script terminates.

In this example:

```
IF EDITTEXT (1) = "" OR EDITTEXT (2) = ""
PERFORM no_response
ELSE
PERFORM launch_app
```

if either edit text field is empty, DCS performs the subroutine no\_response. If neither field is empty, DCS performs the subroutine launch\_app.

---

# INCREMENT

---

## INCREMENT IntVar

The INCREMENT command increases the value of the specified numeric variable by one.

### Arguments

#### IntVar

The IntVar argument specifies the integer variable to be incremented. An integer variable must be specified, not a real variable. Specifying a real variable causes a syntax error.

### Example

In this example:

```
%index = 0
RECORD READ 0 at 0
WHILE NOT EOF ()
BEGIN
DISPLAY (%index, 0) "Record " | STR (%index) | @R0
INCREMENT %index
RECORD READ 0
END
```

the INCREMENT command is used as a counter to display the record number and the contents of the record in the session window.

---

## KERMIT COPY

---

### **KERMIT COPY SourceFile TO DestinationFile WINDOW WinHandle**

Sends the Kermit copy command packet to the remote (server) system. The remote system must support Kermit server mode.



Note: This command does not apply to IBM TN3270 emulations.

#### **Arguments**

##### **SourceFile**

The **SourceFile** argument is a string specifying the file to copy on the remote system and must be a valid file name on that system.

##### **TO DestinationFile**

The **TO** clause specifies the file name for the copy of the **SourceFile**. The clause is composed of the **TO** keyword and the **DestinationFile** argument. The **DestinationFile** argument is a string specifying the file name of the copy of the file on the remote system and must be a valid file name on that system.

##### **WINDOW WinHandle**

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

#### **Comments**

You must specify the Kermit binary file transfer protocol for the session before performing this operation. DCS displays all responses from the remote system in the session window. The remote system must be in Kermit server mode.

If no window handle is specified, the **KERMIT COPY** command is performed upon the active session.

#### **Example**

In this example:

```
KERMIT COPY "OLDFILE.C" TO "NEWFILE.C"
```

a copy of **OLDFILE.C** is created on the remote system and named **NEWFILE.C**.

---

## KERMIT DIRECTORY

---

### **KERMIT DIRECTORY DirectoryName WINDOW WinHandle**

Sends the Kermit directory command to the remote (server) system. The remote system must support Kermit server mode.



Note: This command does not apply to IBM TN3270 emulations.

#### **Arguments**

##### **DirectoryName**

The **DirectoryName** argument is a string specifying the directory name for which the contents will be listed.

##### **WINDOW WinHandle**

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

#### **Comments**

You must specify the Kermit binary file transfer protocol for the session before performing this operation. DCS displays all responses from the remote system in the session window. The remote system must be in Kermit server mode.

If no window handle is specified, the **KERMIT DIRECTORY** command is performed upon the active session.

#### **Example**

In this example:

```
KERMIT DIRECTORY "Documnts"
```

the **KERMIT DIRECTORY** command lists all of the files contained in the **DOCUMNTS** directory on the remote system.

---

## KERMIT ERASE

---

### **KERMIT ERASE** *FileName* **WINDOW** *WinHandle*

Sends the Kermit erase command to a remote system. The Kermit erase command deletes a file from a directory on the remote system. The remote system must support Kermit server mode.

 Note: This command does not apply to IBM TN3270 emulations.

#### **Arguments**

##### **FileName**

The **FileName** argument is a string specifying the name of the file to be deleted from the remote system.

##### **WINDOW WinHandle**

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

#### **Comments**

You must specify the Kermit binary file transfer protocol for the session before performing this operation. DCS displays all responses from the remote system in the session window. The remote system must be in Kermit server mode.

If no window handle is specified, the **KERMIT ERASE** command is performed upon the active session.

#### **Example**

In this example,

```
KERMIT ERASE "FAX.DAT"
```

the **KERMIT ERASE** command deletes the file `FAX.DAT` from the remote system.


---

# KERMIT FINISH

---

## KERMIT FINISH WINDOW WinHandle

The KERMIT FINISH command sends a Kermit finish packet to the remote Kermit server. The remote system must support Kermit server mode.

 Note: This command does not apply to IBM TN3270 emulations.

### Arguments

#### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

### Comments

You must select the Kermit binary file transfer protocol for the session before using this command. DCS displays all responses from the remote system in the session window. The remote system must be in Kermit server mode.

If no window handle is specified, the KERMIT FINISH command is performed upon the active session.

### Example

In this example:

```
SEND "KERMIT"  
KERMIT DIRECTORY  
KERMIT FINISH
```

DCS sends a Kermit finish packet to the remote Kermit server after displaying a directory.



---

# KERMIT FREESPACE

---

## KERMIT FREESPACE WINDOW WinHandle

The KERMIT FREESPACE command displays the number of free bytes on the remote system. The remote system must support Kermit server mode.



Note: This command does not apply to IBM TN3270 emulations.

### Arguments

#### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

### Comments

You must specify the Kermit binary file transfer protocol for the session before performing this operation. DCS displays all responses from the remote system in the session window. The remote system must be in Kermit server mode.

If no window handle is specified, the KERMIT FREESPACE command is performed upon the active session.

### Example

In this example:

```
KERMIT FREESPACE
```

DCS displays the number of free bytes on the remote system in a session window.

---

# KERMIT HELP

---

## KERMIT HELP HelpTopic WINDOW WinHandle

The KERMIT HELP command displays a brief summary of instructions on a topic of the remote system's operation. The remote system must support Kermit server mode.



Note: This command does not apply to IBM TN3270 emulations.

### Arguments

#### HelpTopic

The HelpTopic is a string specifying the name of a topic included in the online help of the remote system.

#### WINDOW WinHandle

The optional WINDOW clause includes the WINDOW keyword and the WinHandle argument. The WinHandle argument is an integer identifying a particular session window.

### Comments

You must specify the Kermit binary file transfer protocol for the session before performing this operation. DCS displays all responses from the remote system in the session window. The remote system must be in Kermit server mode.

If no window handle is specified, the KERMIT HELP command is performed upon the active session.

### Example

In this example:

```
KERMIT HELP "DELETE"
```

DCS displays the help text concerning file deletion on the remote system in a session window.

---

# KERMIT LOGOUT

---

## KERMIT LOGOUT WINDOW WinHandle

The KERMIT LOGOUT command sends a Kermit logout packet to the remote Kermit server. The remote system must support Kermit server mode.



Note: This command does not apply to IBM TN3270 emulations.

### Arguments

#### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

### Comments

You must select the Kermit binary file transfer protocol for the session before performing this operation. The remote system must be in Kermit server mode.

If no window handle is specified, the KERMIT LOGOUT command is performed upon the active session.

### Example

In this example:

```
SET BINARYTRANSFERS KERMIT
FILE SEND BINARY REMOTE "MEMO.TXT"
KERMIT LOGOUT
```

DCS sends a Kermit logout packet to the remote system after completing the file transfer.


---

# KERMIT MESSAGE

---

## **KERMIT MESSAGE ReceiverName MessageText WINDOW WinHandle**

The KERMIT MESSAGE command sends a short message to an account on the remote system. The remote system must support Kermit server mode.

 Note: This command does not apply to IBM TN3270 emulations.

### **Arguments**

#### **ReceiverName**

The ReceiverName is a string specifying a valid user name on the remote system.

#### **MessageText**

The MessageText is the string sent to the user name on the remote system.

#### **WINDOW WinHandle**

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

### **Comments**

You must specify the Kermit binary file transfer protocol for the session before performing this operation. DCS displays all responses from the remote system in the session window. The remote system must be in Kermit server mode.

If no window handle is specified, the KERMIT MESSAGE command is performed upon the active session.

### **Example**

In this example:

```
KERMIT MESSAGE "FARRELLT" "HELLO"
```

DCS sends the message "Hello" to the user FARRELLT on the remote system.

---

# KERMIT NEWDIRECTORY

---

## **KERMIT NEWDIRECTORY DirectoryName WINDOW WinHandle**

The **KERMIT NEWDIRECTORY** command changes the current working directory on the remote system to a new directory. The remote system must support Kermit server mode.



Note: This command does not apply to IBM TN3270 emulations.

### **Arguments**

#### **DirectoryName**

The **DirectoryName** argument is a string specifying the new working directory on the remote system.

#### **WINDOW WinHandle**

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

### **Comments**

You must specify the Kermit binary file transfer protocol for the session before performing this operation. DCS displays all responses from the remote system in the session window. The remote system must be in Kermit server mode.

If no window handle is specified, the **KERMIT NEWDIRECTORY** command is performed upon the active session.

### **Example**

In this example:

```
KERMIT NEWDIRECTORY "LETTERS"
```

the command changes the current working directory to the directory **LETTERS**.

---

# KERMIT RENAME

---

## **KERMIT RENAME OldFileName TO NewFileName WINDOW WinHandle**

The KERMIT RENAME command changes the name of a file on the remote system to a new name. The remote system must support Kermit server mode.



Note: This command does not apply to IBM TN3270 emulations.

### **Arguments**

#### **OldFileName**

The OldFileName argument is a string, specifying the name of the file to change on the remote system.

#### **TO NewFileName**

The TO clause specifies the name to which the file will be renamed. The TO clause is composed of the TO keyword and the argument, NewFileName. The NewFileName is a string argument, specifying the new name for the file on the remote system.

#### **WINDOW WinHandle**

The optional WINDOW clause includes the WINDOW keyword and the WinHandle argument. The WinHandle argument is an integer identifying a particular session window.

### **Comments**

You must specify the Kermit binary file transfer protocol for the session before performing this operation. DCS displays all responses from the remote system in a session window. The remote system must be in Kermit server mode.

If no window handle is specified, the KERMIT RENAME command is performed upon the active session.

### **Example**

In this example:

```
KERMIT RENAME "DATA2A" TO "STCKQUOT"
```

the file DATA2A is renamed STCKQUOT on the remote system.


---

## KERMIT TYPE

---

### **KERMIT TYPE FileName WINDOW WinHandle**

The **KERMIT TYPE** command displays the contents of a file located on the remote system. The remote system must support Kermit server mode.

 **Note:** This command does not apply to IBM TN3270 emulations.

#### **Arguments**

##### **FileName**

The **FileName** argument is a string specifying a file on the remote system.

##### **WINDOW WinHandle**

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

#### **Comments**

You must specify the Kermit binary file transfer protocol for the session before performing this operation. DCS displays all responses from the remote system in a session window. The remote system must be in Kermit server mode.

If no window handle is specified, the **KERMIT TYPE** command is performed upon the active session.

#### **Example**

In this example:

```
KERMIT TYPE "TEXT1"
```

DCS displays the contents of the file named **TEXT1** in the session window.

---

# KERMIT WHO

---

## **KERMIT WHO** *UserName WINDOW WinHandle*

The **KERMIT WHO** command issues the Kermit who command to the remote system. The remote system must support Kermit server mode.



Note: This command does not apply to IBM TN3270 emulations.

### **Arguments**

#### **UserName**

The **UserName** argument is an optional string argument. If it is not included in the command, the command displays the names of all the people logged into the remote system. If the argument is included, the command displays a message indicating the status of a particular person on the remote system.

#### **WINDOW WinHandle**

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

### **Comments**

You must specify the Kermit binary file transfer protocol for the session before performing this operation. DCS displays all responses from the remote system in a session window.

The results of this command vary from system to system. The implementation of the Kermit who command may be different from one system to the next. The remote system must be in Kermit server mode.

If no window handle is specified, the **KERMIT WHO** command is performed upon the active session.

### **Example**

In this example:

```
KERMIT WHO
```

DCS displays a list of all the persons logged into the remote system in the session window.



---

# KEY

---

## KEY Modifier1 Key1 Definition WINDOW WinHandle

The KEY command remaps a key (or key combination) to a new definition. The definition can be a meta key or a string.

### Arguments

#### Modifier1

The optional **Modifier1** argument is specified by one or more of the following keywords: ALT, CTRL, LEFTSIDE, RIGHTSIDE, or SHIFT.

#### Key1

The **Key1** argument specifies a valid key name. Valid keys are letters (A through Z), digits (0 through 9), or one of the key names listed in the table below.

#### Definition

The **Definition** argument can be specified as a string.

#### String

If a string is specified, the **Definition** argument consists of any valid string (literal, variable, function, etc). Thus, you may remap a key to send a string rather than a character. When a remapped key is pressed in an active session window, the string is sent to the remote system. When a remapped key is pressed in an active script or memo window, the string is inserted in the script or memo at the current cursor location.

String	Command
^\$B	BREAK
^\$C	DIAL
^\$D	WAIT DELAY
^\$E	EXECUTE
^\$H	HANGUP
^\$L	LEVEL
^\$P	PERFORM
^\$R	SET RESULT



**Note:** Embedded control characters must begin the string or be the only characters in the string.

---

## KEY, *continued*

---

Key Name	Indicated Key
Add	Plus Key (Num Pad)
Alt	Alternate Key
Back	Backspace Key
Cancel	Cancel Key
Capital	Cap Lock Key
Clear	Clear Key
CtrlKey	Control Key
Decimal	Period Key (Num Pad)
Delete	Delete Key
Divide	Slash Key / (Num Pad)
Down	Down Arrow Key
End	End Key
Escape	Escape (Esc) Key
F1	Function Key One
through	through
F16	Function Key Sixteen
Home	Home Key
Insert	Insert Key
Left	Left Arrow Key
Multiply	Asterisk Key (Num Pad)
Next	Next Key
NumLock	Num Lock Key
NumPad0	Zero Key (Num Pad)
through	
NumPad9	Nine Key (Num Pad)
Pause	Pause Key
Print	Print Key
Prior	Prior Key
Reset	Reset Key
Return	Return (or Enter) Key
Right	Right Arrow Key
ScrLock	Scroll Lock Key
Select	Select Key
Separator	Comma Key (Num Pad)
ShiftKey	Shift Key
Space	Space Bar
Subtract	Minus Key (Num Pad)
Tab	Tab Key
Up	Up Arrow Key
186	Semicolon and Colon Key
187	Equals and Plus Key
188	Comma and Less Than Key
189	Hyphen and Underline Key

---

## KEY, *continued*

---

Key Name	Indicated Key
190	Period and Greater Than Key
191	Slash and Question Mark Key
192	Grave and Tilde Key
219	Left Bracket and Left Brace Key
220	Backslash and Bar Key
221	Right Bracket and Right Brace Key
222	Quote and Double Quote Key

### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

The **WINDOW** clause directs DCS to apply the keyboard remapping to a key map attached to a particular session window.

### Comments

If the **WINDOW** clause is not included in the command, the remapping is applied to the current session.

### Example

In this example:

```
KEY SHIFT G "GOTO"
```

when the [SHIFT]+[G] key combination is pressed, the string GOTO is inserted at the current insertion point.

In this example:

```
KEY SHIFT SPACE ``^$B``
```

when the [SHIFT]+[SPACE] key combination is pressed, a break is sent to the remote system. The host system determines the break length and behavior.

In this example:

```
KEY Back CHR (127)
```

the **CHR** function returns the delete character (the character that the [DELETE] key sends), and the **KEY** command changes the character that the [BACKSPACE] key sends to the delete character.

---

## KEY, *continued*

---

In this example:

```
KEY LEFTSIDE INSERT ``^[2J``
```

this command assigns the escape sequence `^[2J` to the middle cursor cluster [INSERT] key. When VT-100 is the session's current emulation, this character sequence erases the terminal screen.



Also see: Escape sequences for your emulation in the DCS 9 Online Reference or specific Client Option Help.

In this example:

```
KEY RIGHTSIDE HOME ``^[OP``
```

this command assigns the escape sequence `^[OP` to the keypad [HOME] key. When a VT-220 emulation is active for a session, this character sequence makes the [HOME] key act like a [PF1].

In this example:

```
KEY NUMPAD5 ``^G``
```

assigns a `^G` (bell) character to the keypad [5] key.

In this example:

```
KEY G ``^$P `ringtest*ring'``  
WAIT RESUME  
  
*ring  
DISPLAY ``^G``  
RETURN
```

after this script is executed, when the [G] key is pressed, the script will start executing at the far label, `*ring`, in the RINGTEST script.

---

# KEYBOARD

---

## KEYBOARD State WINDOW WinHandle

The **KEYBOARD** command determines whether keystrokes are transmitted to the local or remote system.

### Arguments

#### State

The **State** argument is specified by one of the following keywords:

Keyword	Description
BUFFER	Directs DCS to buffer keystrokes until a <b>KEYBOARD LOCK</b> , <b>KEYBOARD UNLOCK</b> , or <b>KEYBOARD ECHO</b> command is executed.
ECHO	Directs DCS to transmit keystrokes to the local system only.
LOCK	Directs DCS to transmit keystrokes to neither the local nor the remote system.
UNLOCK	Directs DCS to transmit keystrokes to both the local and remote systems.
WAIT	Directs DCS to buffer keystrokes until a <b>KEYBOARD LOCK</b> , <b>KEYBOARD UNLOCK</b> , <b>KEYBOARD ECHO</b> , or <b>WAIT</b> command is executed.

#### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

The **WINDOW** clause directs DCS to change the keyboard state for a particular session.

### Comments

If the **WINDOW** clause is not included in the command, the keyboard state is changed for the active session.

### Example

In this example:

```
LOAD "TRANSET"  
KEYBOARD LOCK  
FILE SEND BINARY "SCRIPT1.DCP"  
KEYBOARD UNLOCK
```

the keyboard is locked while the file transfer is performed.

---

## KEYMAP LOAD

---

### KEYMAP LOAD FileName

The KEYMAP LOAD command loads the keymap file specified by the file name.

#### Arguments

##### FileName

The FileName argument is a string specifying the keymap file to be loaded. The FileName argument must specify a valid file name for your system.

#### Comments

The ERROR function returns TRUE if the specified file does not exist.

To load the default keymap for the emulation, use the KEYMAP RESET command.

---

# KEYMAP RESET

---

## KEYMAP RESET

The KEYMAP RESET command returns the keyboard and keymapping to the default settings for the emulation.

### Arguments

The KEYMAP RESET command takes no arguments.

### Example

```
KEY SHIFT SPACE ``^$B20``  
KEY ALT L ``Login``  
PERFORM ``login.dcp *label``  
KEYMAP RESET
```

In this example, the [SHIFT]+[SPACE] and [ALT]+[L] key combinations are modified. After performing the target script branch, the keyboard is reset and both key combinations return to their default settings.

---

## KEYMAP SAVE

---

### KEYMAP SAVE FileName

The KEYMAP SAVE command saves the current keymap file with a specified file name.

#### Arguments

##### FileName

The FileName argument is a string specifying the keymap file to be saved. The FileName argument must specify a valid file name for your system.

#### Comments

The file cannot be saved to the default keymap; a saved keymap must always be given a specific file name.



---

# LAUNCH

---

## LAUNCH Application Command, ... ContinueScript

The LAUNCH command starts the specified program as a separate application running under Windows.

### Arguments

#### Application

The Application argument is a string specifying the file to launch. The file must have a command extension appropriate to the operating system (such as EXE , COM, or BAT).

#### Command, ...

The optional **Command** argument is a string specifying a list of command line instructions sent to the specified application. The commands must be in the launched application's terms and separated by commas.

#### ContinueScript

The optional **ContinueScript** argument indicates whether DCS should launch the application and continue executing the script, or launch the application and halt the script until the target application closes. It requires one of the following keywords:

Keyword	Description
SYNCHRONOUS	If you include the SYNCHRONOUS keyword in the command, DCS pauses the script while the application is running.
ASYNCHRONOUS	If you include the ASYNCHRONOUS keyword in the command, DCS continues the script after starting the application.

If the **ContinueScript** argument is not included, the script continues after starting the application.

### Comments

To launch the application, you must specify the complete path to the application or list the path in AUTOEXEC.BAT.

---

## LAUNCH, *continued*

---

### Example

This script segment:

```
LAUNCH "EXCEL.EXE"  
IF ERROR (  
    BEGIN  
    DISPLAY "Can't launch Excel",  
    BEEP 1  
    CANCEL  
End
```

starts Excel and then tests whether DCS was able to start it. If Excel did not start, DCS places a message in the active session window, beeps the system bell once, and stops the script.

```
LAUNCH $appname SYNCHRONOUS
```

DCS starts the application and then performs the equivalent of a `WAIT RESUME` command to pause the script. When the target application is no longer running, DCS continues the script with the command following the `LAUNCH` command.

```
LAUNCH $appname ASYNCHRONOUS
```

DCS starts the target application and continues with the script command following the `LAUNCH` command.

---

# LEAVE

---

## LEAVE

The `LEAVE` command branches execution past the last command in the currently executing command block.

### Arguments

The `LEAVE` command takes no arguments.

### Comments

The `LEAVE` command may be executed from within the command block of a `WHILE` or `SWITCH` loop.

### Example

In this example:

```
SET %counter 0
WHILE %counter < 10
BEGIN
RECORD READ 0
IF EOF ( )
LEAVE
DISPLAY (%counter, 0) @R0
INCREMENT %counter
END
DISPLAY "No More Records"
```

DCS is directed to read and display ten records from table zero. If the end of the file is reached before ten records are read, the `LEAVE` command branches execution to the line following the `END` command.

---

## LEVEL

---

### LEVEL KeyLevel

The LEVEL command selects a level on a session toolbar.

#### Arguments

##### KeyLevel

The KeyLevel argument is a numeric (from 1 to 4) specifying the toolbar level.

#### Comments

DCS maintains four levels on a toolbar. Each level contains eight buttons consisting of the title and action.



**Note:** This command does not apply to the AT&T 605 emulation.

#### Example

In this example:

```
FKEYS HIDE
LEVEL 1
@F1 = "Today's Pass^M"
@T1 = "password"
@F2 = "MyName^M"
@T2 = "User ID"
LEVEL 2
@F1 = "Go Favorite^M"
@T1 = "My Favorite"
FKEYS SHOW
```

In this example, DCS hides the session toolbar buttons for the active session window and then configures three buttons: two on level one and one on level two. After the buttons are configured, the script displays the toolbar.

---

# LIBRARY CALL

---

## LIBRARY CALL LibraryReference Procedure (Parameter, ...)

The LIBRARY CALL command branches execution to the specified Dynamic Link Library (DLL) procedure.



Note: The calling convention of the library function being used is determined by the routine name. If the name has an underscore as the first character (“\_”) then the ‘C’ programming language convention is used. Otherwise, the function will be called using the standard WINAPI convention (sometimes referred to as the Pascal convention). It is VERY IMPORTANT that the calling convention be correct or unpredictable results will occur, i.e. the application will crash.

### Arguments

#### LibraryReference

The LibraryReference argument specifies the library containing the procedure. The argument is either a string specifying the name of the file containing the procedure, or a numeric variable containing the reference value returned by the LIBRARY LOAD command. If the DLL referenced is not loaded, the LIBRARY CALL command loads, uses, and then unloads the specified library.

#### Procedure

The Procedure argument is a string specifying the name of the procedure to perform, as it appears in the DLL.

#### (Parameter, ...)

The optional (Parameter, ...) argument specifies one or more strings, numerics, or Booleans, enclosed in parentheses and separated by commas, to be passed to the procedure. The passed parameters should correspond to the parameter types expected by the DLL procedure. Parameters are passed by reference only.

### Comments

When calling DLLs from script, the parameters are passed according to the following rules:

1. All variables are passed by reference.

Also see: Parameter Passing section in **Chapter 1 Introduction**

2. The variable types follow the following syntax:

Type	Variable Prefix in Script	SDK	(C)
Boolean	#	LPINT	(int far *)
Integer	%	LPLONG	(long far *)
Real	!	see below	(double far *)
String	\$	LPSTR	(char far *)

---

## LIBRARY CALL, *continued*

---

LPINT, LPLONG, and LPSTR are defined as shown by the WINDOWS.H file in the Windows SDK. The SDK does not have a definition for double far \*.

3. Pascal or C calling convention is determined by the routine name. If it has an underscore as the first character (“\_”) then it will be called using the C convention. Otherwise, it will be called with the Pascal convention. Please note that the Pascal calling convention is the usual method for calling Windows functions.
4. The return value expects a LONG value. DCS takes the four bytes returned by a function and copies the bytes into the Result system variable.



Also see: Windows Software Development Kit (SDK) for more information on the use and development of Dynamic Link Libraries (DLLs).

TYPEDLIBRARYCALL function

### Example

In this example:

```
LIBRARY LOAD "lib1.DLL" %lib1
;loads Dynamic Link Library
LIBRARY CALL %lib1 "addProc" ($file, %size)
LIBRARY CALL %lib1 "deleteProc" ($file, %size)
LIBRARY CALL "lib2.DLL" "printProc" ($file)
LIBRARY CALL %lib1 "saveProc" ($file)
;above commands perform functions of the library
LIBRARY UNLOAD %lib1
```

the script accesses four procedures in two DLLs with two different methods. Since the script accesses three procedures in LIB1.DLL, the script opens the DLL once with the LIBRARY LOAD command and then uses the DLL with the LIBRARY CALL command. The LIBRARY CALL command refers to the DLL by the variable %lib1. When a script needs to access a DLL many times, this method uses less time since the script does not have to open the DLL repeatedly with the LIBRARY CALL command. Finally, the LIBRARY UNLOAD command takes the DLL out of memory.

When the script uses LIB2.DLL, it is using another method of accessing a DLL. Since the script accesses only one procedure in LIB2.DLL, the script uses only the LIBRARY CALL command once to open (referring to the DLL by the DLL's file name), to access, and then to close the library. This method uses less memory than the first method.

---

## LIBRARY CALL, *continued*

---

The following is a C language function extracted from a DLL:

```
typedef double far *LPDOUBLE
long FAR PASCAL MyFunction (LPINT lpiBoolean,
LPLONG lpiInteger,
LPDOUBLE lpdReal,
LPSTR lpszString)
{
return ((long) *lpiInteger);
}
```

To call the DLL function, you might employ the following script command which:

```
LIBRARY CALL "C:\WINDOWS\MyDLL.DLL" "MyFunction" \
(#Boolean, %Integer, !Real, $String)
%r = Ord (Substr (Result (), 1,1) + \
Ord (Substr (Result (), 2,1) * 0x100 + \
Ord (Substr (Result (), 3,1) * 0x10000+ \
Ord (Substr (Result (), 4,1) * 0x1000000
```

calls the DLL function and passes it four parameters (#Boolean, %Integer, !Real, \$String). The DLL returns a LONG value, which is placed in DCS's result system variable. The **RESULT** function makes the return value of the DLL available to the script. The **SUBSTR** function extracts the first four digits of the string representation of the number. The **ORD** function then converts the substring to an integer. The integer is then placed in the integer variable %r.

---

## LIBRARY LOAD

---

### **LIBRARY LOAD LibraryName LibraryReference**

The LIBRARY LOAD command loads the specified Dynamic Link Library (DLL) into memory. It may also return a numeric value which the LIBRARY CALL and LIBRARY UNLOAD commands can use to reference the DLL.

#### **Arguments**

##### **LibraryName**

The LibraryName argument specifies the name of the file containing the desired DLL. It must specify a valid file name for your system.

##### **LibraryReference**

The optional LibraryReference argument is a numeric variable into which DCS stores a numeric reference value returned by the LIBRARY LOAD command. All LIBRARY CALL and LIBRARY UNLOAD commands may then use this reference variable to refer to the loaded library. If it is not included, all references to the loaded library must be made with a string specifying the file name of the library.

If the number stored in the variable is less than zero or a small (2 digit) number, DCS was not able to load the library.

#### **Comments**

This command allows a commonly used DLL to be loaded before multiple calls are made to the routines it contains. Normally, the LIBRARY UNLOAD command is used when the DLL is no longer needed. You may load up to three DLLs concurrently.

#### **Example**

See the LIBRARY CALL command.



---

## LIBRARY UNLOAD

---

### LIBRARY UNLOAD *LibraryReference*

The LIBRARY UNLOAD command removes the specified Dynamic Link Library (DLL) from memory.

#### Arguments

##### *LibraryReference*

The *LibraryReference* argument specifies the library to be unloaded. It is either a string specifying the name of the library file, or a numeric variable containing the reference value returned by the LIBRARY LOAD command.

#### Example

See the LIBRARY CALL command.

---

# LINENUMBERS

---

## LINENUMBERS

The LINENUMBERS command is a compiler directive which directs DCS to include numbers in a compiled script. The numbers indicate the line where a command in a script text file is located.

### Arguments

The LINENUMBERS command takes no arguments.

### Comments

The LINENUMBERS command takes effect only when DCS compiles a script. The line number information in a compiled script may help you debug a script. When you compile a script with the LINENUMBERS command, and if you have included the DEBUG and SHOW commands in the script, the DEBUG command will write the line number information to a debugging file, and any script error dialogs will display the line number of the offending script command rather than the command index.

The LINENUMBERS command does not affect the status of the **Include Line Numbers** option in the **Compile** dialog.

See also the COMPILE command and the DEBUG command.

### Example

When you compile a script containing these commands:

```
LineNumbers
Debug "DBG0131.TXT"
Show
.
.
.
```

DCS will automatically place line numbers in the compiled script (you do not need to check the Include Line Numbers check box in the Compile dialog box). When the script runs, it creates the file DBG0131.TXT and writes each executing command and function to the file. The script precedes each command it places in the file with the command's line number.

```
LineNumbers
Compile "MYSCRIPT.DCP"
.
.
.
```

This script segment causes DCS to compile MYSCRIPT.DCP. As DCS compiles the script, DCS includes line number information. This script segment is similar to checking (enabling) the **Include Line Numbers** option before you start the compilation of a script.

---

# LOAD

---

## LOAD FileName

The LOAD command loads the session file specified by the optional file name.

### Arguments

#### FileName

The optional **FileName** argument is a string specifying the session file to be loaded. The **FileName** argument must specify a valid file name for your system.

If you specify the **FileName** argument as a question mark (?), DCS will display the **Open** dialog, allowing you to select a session file during script execution.

If you do not include the optional **FileName** argument, DCS loads a new session file containing the default settings.

### Comments

The ERROR function returns TRUE if the specified file does not exist.

Since only session files can be loaded using the LOAD command, you do not need to include the SES file extension in the **FileName** argument.

The LOAD command does not execute if the Boolean argument of the SET TERM CLOSE command evaluates to FALSE. When the SET TERM CLOSE command has a false argument, you may not close an open session window. Opening or loading a session file implies that you want to close an open session file or window.

### Example

In this example:

```
LOAD "SESSNAME"  
IF ERROR ()  
BEGIN  
  DISPLAY "Could not load session file"  
  CANCEL  
END  
PERFORM login_sequence
```

DCS attempts to load the session file SESSNAME before performing the login sequence. If the session file cannot be loaded, DCS displays an error message in the session window and cancels execution.

---

## LOGTOFILE

---

### LOGTOFILE FileName WINDOW WinHandle

The LOGTOFILE command activates the option to log incoming data to a file. A second instance of the command directs DCS to stop logging data and close the file.

#### Arguments

##### FileName

The FileName argument is a string specifying the name of the file to which the incoming data will be logged. You must enter a file name; DCS will not prompt for one and if no name is specified no log will be written.

##### WINDOW WinHandle

The optional WINDOW clause includes the WINDOW keyword and the WinHandle argument. The optional WinHandle argument is an integer identifying a particular session window.

If no window handle is specified, this command applies to the active session window.

#### Comments

This command has the same effect as selecting **Session > Log Incoming Data > Log to File**.

Use the command a second time to stop logging incoming data and close the file.

Use the FILE PAUSE and FILE RESUME commands to pause and resume file logging.

---

# MENU

---

**MENU**  
  Option  
  Option...  
**MENU END**

The **MENU** command is used to reconfigure a DCS application menu, allowing for the addition or restriction of available menu options. The **MENU** command is followed by one or more menu options. You must follow the **Option** arguments with a **MENU END** command, indicating the end of the menu definition.

## Arguments

### Option

The **Option** argument is any of the following menu commands:

          ITEM          POPUP          SEPARATOR

If the **Option** argument is not included, only the DCS application control menu is shown. (The control menu is also called the System menu and is located in the upper left hand corner of the window.)

## Comments

DCS restores its standard menu bar upon executing a **MENU CANCEL** command, or upon the end of script execution.

## Example

See each individual menu option command.

---

## MENU CANCEL

---

### MENU CANCEL

The MENU CANCEL command removes the menu bar created by a script, restoring DCS's standard menu bar.

#### Arguments

The MENU CANCEL command takes no arguments.

#### Comments

DCS restores its standard menu bar upon executing a MENU CANCEL command, or upon the end of script execution.

#### Example

In this example:

```
MENU
  POPUP "File" SYSTEM 2
  POPUP "Edit" SYSTEM 4
  POPUP "Session" SYSTEM 11
  POPUP "View" SYSTEM 6
  POPUP "Windows" SYSTEM 10
  POPUP "Help" SYSTEM 9
MENU END
PERFORM setup
MENU CANCEL
PERFORM receive_files
```

the menu bar definition removes the **Tools** and **Script** menus from the menu bar. The MENU CANCEL command restores these options before the receive\_files routine is performed.

---

## MENU DELETE ITEM

---

### MENU DELETE ITEM IntPopup Intltem

The MENU DELETE ITEM command removes an item from a popup menu.

#### Arguments

##### IntPopup

The IntPopup argument is an integer expression indicating the popup menu from which the item will be deleted.

The first popup menu on the left side of the menu bar is popup menu 1 (one).

##### Intltem

The Intltem argument is an integer expression indicating the position of the item to be deleted. The first item in the popup menu is item 1 (one). When counting or numbering items, be sure to count both menu items and separators.

#### Comments

If there is no menu defined via the MENU command, the ERROR function returns TRUE.

Once an item is deleted, all items following the deleted item move up one position in the popup menu. For example, if a popup menu contains five items and item three is deleted, items four and five immediately move up one position in the menu, becoming items three and four.

#### Example

In this script segment:

```
MENU DELETE ITEM 2 3
MENU DELETE ITEM 2 3
```

the third and fourth menu selections (**Send** and **Clear Screen**) are removed from the second popup menu (**Edit**). The first line of the script removes from the **Edit** menu (popup menu 2) the **Send** selection (item 3). The **Clear Screen** item in the **Edit** menu immediately moves up one position and becomes the third menu selection. Therefore, to remove the **Clear Screen** selection, the second line of the script again removes from the third menu selection from the second popup menu .

---

## MENU DELETE POPUP

---

### MENU DELETE POPUP IntPopup

The MENU DELETE POPUP command removes a popup menu from the menu bar.

#### Arguments

##### IntPopup

The IntPopup argument is an integer expression indicating the popup menu to be deleted.

The first popup menu on the left side of the menu bar is popup menu 1 (one).

#### Comments

If there is no menu defined via the MENU command, the ERROR function returns TRUE.

Once a popup menu is deleted, all popup menus following the deleted menu move left one position on the menu bar. For example, if a popup menu contains five popup menus and menu three is deleted, menus four and five immediately move left one position on the menu bar, becoming popup menus three and four.

#### Example

In this script segment:

```
MENU DELETE POPUP 3  
MENU DELETE POPUP 3
```

the third and fourth popup menus, **Session** and **Script**, are removed. The first line of the script removes the **Session** menu (popup menu 3) from the menu bar. The **Script** menu immediately moves one position to the left and becomes the third popup menu on the menu bar. Therefore, the second line of the script again removes the third popup menu, **Script**.



---

## MENU INSERT ITEM

---

### MENU INSERT ITEM *IntPopup IntItem StrText Enabled Checked Command*

The MENU INSERT ITEM command adds an item to a popup menu.

#### Arguments

##### IntPopup

The *IntPopup* argument is an integer expression indicating the popup menu into which the item will be inserted.

The first popup menu on the left side of the menu bar is popup menu 1 (one).

##### IntItem

The *IntItem* argument is an integer expression indicating the item's position in the menu. The first item in the popup menu is item 1 (one). When counting or numbering items, be sure to count both menu items and separators.

##### StrText

The *StrText* argument is string indicating the name of the menu item. To assign an [Alt] + [*key*] combination to the menu item, precede the letter of the [*key*] with an ampersand (&). The letter appears underlined in the text of the menu item. For example, the string `Actio&n` underlines the letter 'n' and the menu item is selected with the [Alt] + [N] key combination.

##### Enabled

The optional **Enabled** argument uses the keyword `ENABLED` to indicate that the menu item is active (not dimmed), and can be selected. If this keyword is not specified in the command, the menu item will not be enabled.

##### Checked

The optional **Checked** argument uses the keyword `CHECKED` to indicate the menu item is checked and any command associated with the menu item is active. If this keyword is not specified in the command, the menu item will appear unchecked.

##### Command

The *Command* argument is a command, or block of commands, indicating the DCS command or function executed when the menu item is selected or checked by the user.

#### Comments

If there is no menu defined via the `MENU` command, the `ERROR` function returns `TRUE`.

Once an item is inserted, all items following the inserted item move down one position in the popup menu. For example, if a popup menu contains five items and item is inserted as item three, existing items three, four, and five immediately move down one position in the menu, becoming items four, five, and six.

---

## MENU INSERT ITEM, *continued*

---

### Example

```
MENU INSERT ITEM 3 4 "Library of Congress" ENABLED LOAD  
"LIBCONG"
```

In this example, the menu item "Library of Congress" is added as the fourth menu item of the third popup menu. It appears in the menu as enabled. When selected, DCS loads the session file LIBCONG.

---

## MENU INSERT POPUP

---

### MENU INSERT POPUP IntPopup StrText

The MENU INSERT POPUP command adds a popup menu to the menu bar.

#### Arguments

##### IntPopup

The IntPopup argument is an integer expression indicating the popup menu to be added to the menu bar.

The first popup menu on the left side of the menu bar is popup menu 1 (one).

##### StrText

The StrText argument is string indicating the name of the popup menu. To assign an [Alt] + [key] combination to the menu item, precede the letter of the [key] with an ampersand (&). The letter will appear underlined in the text of the popup menu. For example, the string Li&braries will underline the letter 'b' and the menu item is selected by the [Alt] + [B] key combination.

#### Comments

If there is no menu defined via the MENU command, the ERROR function returns TRUE.

Once a popup menu is added to the menu bar, all menus following the added menu move right one position on the menu bar. For example, if the menu bar contains five popup menus and a popup menu is added as menu three, existing menus three, four, and five immediately move right one position on the menu bar, becoming menus four, five, and six.

#### Example

In this example:

```
MENU INSERT POPUP 4 "Online Li&braries"  
MENU INSERT ITEM 4 1 "NYP&L" ENABLED LOAD "NYPL"  
MENU INSERT ITEM 4 2 "Univ. of &Houston" ENABLED LOAD  
"UOFH"  
MENU INSERT ITEM 4 3 "Univ. of &Michigan" ENABLED LOAD  
"UOFM"  
MENU INSERT ITEM 4 4 "Library of Con&gress" ENABLED LOAD  
"LIBCONG"
```

the popup menu "Online Libraries" is added as the fourth popup menu on the menu bar. It can be selected with the [Alt] + [B] key combination. Four menu items are also added to this menu, which displays when the new menu is selected.

---

## (MENU) ITEM

---

### ITEM Text Enabled Checked Command, ...

The (MENU) ITEM command adds a menu item to the most recently added popup menu.

#### Arguments

##### Text

The **Text** argument is a string specifying the characters to display on the menu item.

##### Enabled

The optional **Enabled** argument is specified by one of the following keywords:

Keyword	Description
ENABLED	Directs DCS to allow selection of the specified item
DISABLED	Directs DCS to disallow selection of the specified item
GRAYED	Directs DCS to disallow selection of the specified item and display the item in gray (or dimmed) type

##### Checked

The optional **Checked** argument is specified by one of the following keywords:

Keyword	Description
CHECKED	Directs DCS to place a check mark before the specified item in the menu
UNCHECKED	Directs DCS to remove a check mark preceding the specified item in the menu

##### Command, ...

The **Command** argument specifies a logical command (either a single command or a command block). Selecting the menu item executes the logical command.

##### Comments

If a character in the **Text** argument is preceded by an ampersand (&), the character appears underlined. The menu or menu item can then be selected by pressing the [ALT] key and the key of the underlined character simultaneously.

You may include a maximum of 200 items in a menu definition.

---

## (MENU) ITEM, *continued*

---

### Example

In this script segment:

```
MENU
  POPUP "&Names"
  ITEM "&Add..." PERFORM add_name
MENU END
WAIT RESUME
```

the menu item **Add...** performs the `add_name` subroutine. Through the keyboard, you can access the **Add...** menu item by pressing the `[ALT]+[A]` key combination.

---

## (MENU) POPUP

---

### POPUP Text SYSTEM Pos

The (MENU) POPUP command adds a popup menu to the menu bar.

#### Arguments

##### Text

The **Text** argument is a string specifying the name of the popup menu to display.

##### SYSTEM Pos

The optional **SYSTEM** clause takes the added popup menu from DCS default defined menus. If the clause is not included, the added popup menu bar contains no items from the system menu bar. The **Pos** argument is a numeric specifying the predefined menu as defined below.

#### Comments

The defined SYSTEM menus are:

System	Menu	Active window type*
SYSTEM 1	File	MAIN
SYSTEM 2	File	SESSION, SCRIPT, MEMO
SYSTEM 3	Edit	MEMO, SCRIPT
SYSTEM 4	Edit	SESSION
SYSTEM 5	Script	MAIN, MEMO, SCRIPT, SESSION
SYSTEM 6	View	MAIN, MEMO, SCRIPT, SESSION
SYSTEM 7	Tools	MAIN, SESSION
SYSTEM 8	Tools	MEMO, SCRIPT
SYSTEM 9	Help	MAIN, MEMO, SCRIPT, SESSION
SYSTEM 10	Window	MEMO, SCRIPT, SESSION
SYSTEM 11	Session	SESSION
SYSTEM 12	Insert	MEMO

- \* The visible menus vary depending on which type of window is active (in focus). MAIN refers to the DCS application window. SESSION refers to session windows, MEMO to memo windows, and SCRIPT to a script editor window.

---

## (MENU) POPUP, *continued*

---

### Example

In this script segment:

```
MENU
  POPUP "File" SYSTEM 1
  POPUP "Edit" SYSTEM 3
MENU END
```

a custom menu bar is defined containing only DCS's File and Edit menus.

---

## (MENU) SEPARATOR

---

### SEPARATOR

The (MENU) SEPARATOR command adds a menu item separator, displayed as a horizontal line, to the popup menu.

### Arguments

The SEPARATOR command takes no arguments.

### Example

In this script segment:

```
MENU
  POPUP "&Names"
  ITEM "&Add ... " PERFORM add_name
  ITEM "&Delete ... " PERFORM delete_name
  SEPARATOR
  ITEM "Di&splay" PERFORM display_name
MENU END
WAIT RESUME
```

when **Add** is selected on the **Names** menu, the `add_name` routine is performed. When **Delete** is selected, the `delete_name` routine is performed. When **Display** is selected, the `display_name` routine is performed.



---

# MENU UPDATE

---

## MENU UPDATE Popup Item Text Enabled Checked

The MENU UPDATE command updates a previously defined menu option, or group of options.

### Arguments

#### Popup

The **Popup** argument is a numeric specifying which popup menu is to be updated. The control menu is specified by the integer zero.

#### Item

The optional **Item** argument is an integer specifying a menu item to update within a popup. The first item is considered item one. When you are numbering items in a menu, include both menu items and separators; however, you may only update menu items, not separators.

If you do not include the **Item** argument, the **Text** argument becomes the text for the menu name, and the **Enabled** and **Checked** arguments change the state of each menu item.

#### Text

The optional **Text** argument is a string specifying the replacement text for a menu or item. If you do not included the **Item argument**, the contents of the **Text** argument replaces the name of the menu.

#### Enabled

The optional **Enabled** argument is specified by one of the following keywords:

Keyword	Description
ENABLED	Directs DCS to allow selection of the specified item
DISABLED	Directs DCS to disallow selection of the specified item
GRAYED	Directs DCS to disallow selection of the specified item and display the item in gray (or dimmed) type.

If the **Item** argument is not specified, the **Enabled** argument applies the same state to all items in the menu.

#### Checked

The optional **Checked** argument is specified by one of the following keywords:

Keyword	Description
CHECKED	directs DCS to place a check mark before the specified item in the menu
UNCHECKED	directs DCS to remove a check mark preceding the specified item in the menu.

If the **Item** argument is not specified, the **Checked** argument applies the same state to all items in the menu.

---

## MENU UPDATE, *continued*

---

### Comments

If there is no menu defined via the MENU command, the ERROR function returns TRUE.

### Example

In this script segment:

```
MENU
  POPUP "File" system 1
  POPUP "Script" system 5
  SEPARATOR
  ITEM "checkme" Unchecked DISPLAY "Checkme!"
MENU END
WAIT DELAY "9"
MENU UPDATE 2 7 CHECKED
WAIT DELAY "9"
```

the selection `checkme` is added to the **Script** menu. After nine seconds, the script checks (selects) the `checkme` item.

---

# NOSHOW

---

## NOSHOW

The NOSHOW command is a compiler directive which cancels the SHOW command and removes the display of a script command from the status bar in the DCS application window.

### Arguments

The NOSHOW command takes no arguments.

### Comments

The NOSHOW command can be used in conjunction with the SHOW and DEBUG commands to control which command lines are written to a debug file.



Also see: SHOW command

DEBUG command

---

## PARSE

---

### PARSE String StringVar1 Keyword StringVar2

The PARSE command locates the keyword in the specified string. If the string contains the keyword, all characters preceding the keyword are stored in the first string variable, and all characters following the keyword are stored in the second string variable.

#### Arguments

##### String

The String argument contains the string in which DCS searches for the keyword.

##### StringVar1

The StringVar1 argument specifies a string variable in which DCS stores all characters in the string preceding the keyword.

##### Keyword

The Keyword argument specifies the string for which to search.

##### StringVar2

The StringVar2 argument specifies a string variable in which to store all characters in the string that appear after the keyword.

#### Comments

The ERROR function returns TRUE if the string does not contain the keyword.

#### Example

In this example:

```
PARSE 'EVANSTON, IL 60202' $city ',' $rest
```

the keyword is a comma (.). DCS assigns the string “EVANSTON” to the variable \$city, and assigns the string “IL 60202” to the variable \$rest.

---

## PARSE, *continued*

---

In this example:

```
#Found = FALSE
$hnd = hwndlist (-1) ;child handles
WHILE $hnd != ""
BEGIN
parse $hnd $h " , " $hnd
%thnd = num ($h)
IF WNDCLASS (%thnd) = 1
BEGIN
#Found = TRUE
LEAVE
END
END
IF #Found
DISPLAY "Session window Handle = " | str (%thnd) | "^M"
ELSE
DISPLAY "Session window Handle not found. ^M"
```

the list of open child window handles is returned and parsed into individual handles. DCS determines the handle of the session window.

In this example:

```
$str1=@r1.1
PARSE $str1 $str2 "break" $str3
@r1.2=$str2
@r1.3=$str3
```

In this example, two table variables (@r1.2 and @r1.3) provide the strings that precede and follow, respectively, the keyword ("break") that is searched for in a string supplied by the first table variable (@r1.1).

---

# PERFORM

---

## PERFORM Target (Parameter, ...)

The PERFORM command causes script execution to branch to the specified target. Execution of the current script resumes at the command following the PERFORM command when the target script executes a RETURN command. When used in this way, the PERFORM and RETURN commands define a subroutine.

### Arguments

#### Target

The **Target** argument specifies either a near (usually a routine in a local script) or a far target (usually another script or a routine in another script).

#### (Parameter, ...)

The optional **(Parameter, ...)** argument specifies the parameters to be passed to the routine identified by the **Target** argument. It consists of one or more strings, numerics, or Booleans.

### Comments



Also see: **Parameter Passing in Chapter 1 Introduction**

### Example

In this example:

```
PERFORM CompileAll ("c:\dcseries\script\*.DCP", TRUE)
.
.
.
*CompileAll ($Scripts, #Make)
ARGUMENTS ($Source)
$Source = ROUTE ($Scripts)
WHILE NOT ERROR ()
BEGIN
  COMPILE $Source #Make
  $Source = NEXT ()
END
RETURN
```

DCS creates a routine that performs a batch compile of all script files in the current directory. The PERFORM command passes two parameters by value to the routine \*CompileAll.

---

## PRINT CANCEL

---

### PRINT CANCEL

The PRINT CANCEL command terminates the active print jobs.

#### Arguments

The PRINT CANCEL command takes no arguments.

#### Comments

Executing the PRINT CANCEL command does not close the currently open print channel.

---

## PRINT CLOSE

---

### PRINT CLOSE

The PRINT CLOSE command closes the open print channel, terminating any active print jobs.

#### Arguments

The PRINT CLOSE command takes no arguments.

#### Example

See the PRINT OPEN command.



---

# PRINT FILE

---

## PRINT FILE FileName LF

The PRINT FILE command prints the specified file.

### Arguments

#### FileName

The FileName argument is a string specifying the file to print. The FileName argument must specify a valid file name for your system.

#### LF

The optional LF keyword adds line feeds to all outgoing carriage returns. If it is not included, carriage returns are printed as carriage returns only.

### Comments

The file is printed according to the active parameters of the open print channel. If no print channel is currently open, the PRINT FILE command automatically opens a print channel and uses the default print parameters.

The ERROR function returns TRUE if the file does not exist, or if DCS cannot open the print channel and print the file.

### Example

In this example:

```
FILE RECEIVE BINARY "MYFILE.TXT"  
PRINT FILE "MYFILE.TXT"
```

the newly received file, MYFILE.TXT, is printed.

---

## PRINT FONT

---

### **PRINT FONT Font Point**

The PRINT FONT command changes the active print font.

#### **Arguments**

##### **Font**

The optional **Font** argument is a string specifying a valid font face for your printer. If it is not included, the current font is used.

##### **Point**

The optional **Point** argument is a numeric specifying a valid point size for the specified font. If it is not included, the current point size is used.

#### **Example**

See the PRINT OPEN command.

---

## PRINT NEWLINE

---

### PRINT NEWLINE

The PRINT NEWLINE command sends a carriage return and line feed to the printer.

#### Arguments

The PRINT NEWLINE command takes no arguments.

#### Example

In this example:

```
PRINT OPEN
PRINT STRING DATE () | ' at ' | TIME ()
PRINT NEWLINE
PRINT TERMINAL ON
PERFORM get_info
PRINT TERMINAL OFF
PRINT CLOSE
```

DCS uses the PRINT NEWLINE command to print a blank line between the header (containing the data and time) and the incoming data.

---

## PRINT NEWPAGE

---

### PRINT NEWPAGE

The PRINT NEWPAGE command sends a form feed to the printer.

#### Arguments

The PRINT NEWPAGE command takes no arguments.

#### Example

In this example:

```
PRINT OPEN
PRINT FONT 'helv' 12
PRINT FILE 'script1.dcp'
PRINT NEWPAGE
PRINT FILE 'script2.dcp'
PRINT NEWPAGE
PRINT FILE 'script3.dcp'
PRINT CLOSE
```

DCS sends a form feed between each file printed.

---

# PRINT OPEN

---

## PRINT OPEN PORT Port DRIVER Driver TYPE Type ABORT

The PRINT OPEN command opens a print channel. DCS must open a print channel before it may execute any other PRINT command (except the PRINT FILE command).

### Arguments

#### PORT Port

The optional **PORT** clause allows you to specify the printer port. The **Port** argument is a string specifying a valid printer port. If the **PORT** clause is not included, DCS uses the default port.

#### DRIVER Driver

The optional **DRIVER** clause allows you to specify the printer driver. The **Driver** argument is a string specifying any installed printer driver. If the **DRIVER** clause is not included, DCS uses the default driver.



Note: UNC standards must be used when specifying a network printer. For example, “\\primeserver\financehp” denotes use of the printer called “financehp” on the “primeserver” system. Notice that four backslashes are used to indicate the server name. When two backslashes are used, DCS interprets them as a single backslash which indicates that a file name follows. The first two backslashes are required to indicate that a system name follows.

#### TYPE Type

The optional **TYPE** clause allows you to specify the printer type. The **Type** argument is a string specifying a valid printer type for the specified driver. If the **TYPE** clause is not included, DCS uses the default type.

#### ABORT

The optional **ABORT** keyword directs DCS to display the Windows **Abort** dialog while printing. This allows you to cancel a print job in progress.

### Example

In this example:

```
PRINT OPEN PORT "LPT1:"
PRINT FONT "helv" 10
PRINT STYLE BOLD
PRINT STRING "MYSCRIPT AS OF #" | DATE ()
PRINT NEWLINE
PRINT STYLE NORMAL
PRINT FILE "MYSCRIPT.DCP"
PRINT CLOSE
```

a print channel is opened, a font is specified, and a header is printed. After MYSCRIPT.DCP is printed, the print channel is closed.

---

## PRINT STRING

---

### **PRINT STRING String**

The PRINT STRING command prints the contents of the specified string according to active print parameters in the open print channel.

#### **Arguments**

##### **String**

The String argument specifies the string to be printed.

#### **Example**

See the PRINT OPEN command.

---

## PRINT STYLE

---

### PRINT STYLE Attribute ...

The PRINT STYLE command sets the print character attributes.

#### Arguments

##### Attribute ...

The Attribute argument is specified by one or more of the following keywords:

Keyword	Description
BOLD	This keyword prints characters boldface type.
ITALIC	This keyword prints characters in italic type.
NORMAL	This keyword directs DCS to eliminate all previously defined attributes and prints characters according to the default style. DCS ignores all attributes specified after NORMAL in a PRINT STYLE command.
QUALITY	This keyword prints characters in letter quality type.
STRIKEOUT	This keyword prints each character with a horizontal rule drawn through the middle of the character.
UNDERLINE	This keyword prints each character as underscored with a horizontal rule.

#### Example

See the PRINT OPEN command.

---

## PRINT TABS

---

### PRINT TABS Width

The PRINT TABS argument specifies the printed tab width.

#### Arguments

##### Width

The Width argument is a numeric specifying the number of spaces per tab. The maximum value of the Width argument is 20.

#### Example

In this example:

```
PRINT OPEN
PRINT STYLE 'courier' 12
PRINT TABS 5
PRINT FILE 'script1.dcp'
PRINT CLOSE
```

the PRINT TABS command specifies a printed tab width of five spaces.



---

# PRINT TERMINAL

---

## PRINT TERMINAL State WINDOW WinHandle

The PRINT TERMINAL command redirects the data appearing in a session window to the printer.



Note: This command does not apply to IBM TN3270 emulations.

### Arguments

#### State

The optional **State** argument is specified by one of the following keywords:

Keyword	Description
ON	This keyword directs DCS to send all data received in the session window to the printer.
OFF	This keyword directs DCS not to send data received in the session window to the printer.
PAUSE	This keyword directs DCS to suspend the sending of data received in the session window to the printer until a PRINT TERMINAL RESUME command is executed.
RESUME	This keyword resumes the PRINT TERMINAL ON command suspended by a PRINT TERMINAL PAUSE command.

If the **State** argument is not included, the PRINT TERMINAL command directs DCS to toggle the current state between ON and OFF.

#### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

### Comments

The file is printed according to the active parameters of the open print channel. If no print channel is currently open, the PRINT TERMINAL command automatically opens a print channel and uses the default print parameters.

If a window handle is not included, the active session window is printed.

### Example

In this example:

```
PERFORM login
PRINT TERMINAL ON
PERFORM get_prices
PRINT TERMINAL OFF
```

DCS captures the communications session and sends it to the printer from the moment the PRINT TERMINAL ON command is issued until the PRINT TERMINAL OFF command executes.

---

## QUIT

---

### QUIT

The QUIT command stops all script execution and terminates the DCS application.

#### Arguments

The QUIT command takes no arguments.

#### Example

In this example:

```
DIALOG 'System Coordinator'  
BUTTON 'Launch Excel'  
BEGIN  
PERFORM launch_xl  
RESUME  
END  
BUTTON 'Quit Excel'  
BEGIN  
PERFORM dde_quit  
RESUME  
END  
BUTTON 'Quit Script' CANCEL  
BUTTON 'Quit DCS' QUIT  
DIALOG END  
WAIT RESUME
```

the user is given the choice of either stopping script execution or quitting DCS.

---

## RECORD FORMAT

---

### **RECORD FORMAT Lines (Row1 Col1) ... (Rown Coln) WINDOW WinHandle**

The RECORD FORMAT command defines a record template the RECORD SCAN command will use. This template specifies the relative position of data fields in the session window, and in which table to place the scanned data.

#### **Arguments**

##### **Lines**

The **Lines** argument is an integer specifying the number of terminal lines between the beginnings of consecutive data records (from the first line of record one to the first line of record two, etc.).

##### **(Row1 Col1) ... (Rown Coln)**

The optional **(Row1 Col1) ... (Rown Coln)** field definitions specify the relative positions in the record template of the desired data fields for each field in a record. Each **Row** coordinate is an integer specifying the row of the starting position of desired data field, where the first row in the record template is row zero. Each **Col** coordinate is a numeric specifying the column of the starting position of desired data field, where the first column is column zero.

##### **WINDOW WinHandle**

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

The **WINDOW** clause directs DCS to define a record template for a particular session window.

#### **Comments**

If the **WINDOW** clause is not included in the command, the record template is defined for the current session.

Since the length of each field is specified by the table definition, the RECORD FORMAT command needs only to specify the starting position, not the length, of each field. You may specify a maximum of 255 field definitions.

If no field definitions are included, the RECORD SCAN command extracts field data from the session window using an auto-scan algorithm in which DCS locates fields by searching for columns delimited by tabs (two or more consecutive spaces).

#### **Example**

See the RECORD SCAN command.

---

## RECORD READ

---

### RECORD READ Table AT Record AT Position LENGTH Bytes

The RECORD READ command reads data from the specified table to the corresponding record buffer. Data contained in a table cannot be directly accessed; one record at a time is accessed through the corresponding record buffer variable, @R*n*, where *n* is a table number.

#### Arguments

##### Table

The Table argument is a numeric specifying the table containing the data.

##### ▼ Structured Tables

###### AT Record

The optional **AT** clause allows random access of records by setting the read position to the specified record number before performing the RECORD READ command. The **Record** argument is a numeric specifying the read position, where the first record is record zero. If the **AT** clause is not included, the RECORD READ command begins reading at record zero, and successive reads are performed sequentially.

After a record read is performed for a structured table, the read pointer is positioned at the next record.

##### ▼ Text Tables

###### AT Position

The optional **AT** clause allows random access of the data blocks (not text lines) by setting the read position to the specified file position before performing the RECORD READ command. The **Position** argument is a numeric specifying the read position, where the first record is record zero. If the **AT** clause is not included, the RECORD READ command begins reading at record zero, and successive reads are performed sequentially.

After a record read is performed for a text table, the read pointer is positioned at the next character (or line).

##### LENGTH Bytes

The optional **LENGTH** clause directs DCS to input a data block of the specified number of bytes (up to 254). The **Bytes** argument is a numeric specifying the number of bytes to be included in each data block. If the **LENGTH** clause is not included, the length of the data block read is determined by a delimiter (a carriage return character followed by a line feed character).

---

## RECORD READ, *continued*

---

### Comments

The EOF function is set to TRUE if an attempt is made to read beyond the end of the file.

### Example

In this example:

```
TABLE DEFINE 0 FIELDS CHAR 10 CHAR 5 REAL 5
TABLE LOAD 0 FROM "DATA" AS SYLK
SET %1 0
RECORD READ 0 at 0
WHILE NOT EOF ()
BEGIN
DISPLAY (%1, 0) @R0
INCREMENT %1
RECORD READ 0
END
```

DCS reads all records from table 0 (zero) and displays them in the session window until it reaches the end of the file.

In this example:

```
TABLE DEFINE 0 TEXT "MYDATA"
RECORD READ 0 AT 200 LENGTH 254
DISPLAY @R0
```

DCS reads 254 bytes of data from the file MYDATA, starting at position 200, then displays the data in the session window.

---

## RECORD SCAN

---

### RECORD SCAN Table WINDOW WinHandle

The RECORD SCAN command, in conjunction with the RECORD FORMAT and SELECTION commands, extracts data from the session window and places it directly into a predefined table structure. This entire process requires three steps: first, use the RECORD FORMAT command to define a virtual record template for the data; second, use the SELECTION command to specify the region of the session window containing the data; and third, use the RECORD SCAN command to transfer the data to the desired table.

#### Arguments

##### Table

The **Table** argument is an integer specifying the table into which DCS stores the data. DCS does not clear the existing contents of the table before adding new data.

##### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

The **WINDOW** clause directs DCS to scan a particular session window.

#### Comments

If the placement of data in the session window is the same for several records on one or more screens, you can use the RECORD SCAN command repeatedly without respecifying the record template or selection range. Using the RECORD SCAN command is a faster and more convenient method of acquiring structured terminal data than using the COLLECT and PARSE commands.

If the number of lines within the selection range exceeds the number of lines specified by the Lines argument of the RECORD FORMAT command, DCS copies a record to the table for each group of lines.

If no field definitions are specified by the RECORD FORMAT command, the RECORD SCAN command extracts field data from the session window using an auto-scan algorithm in which DCS locates fields by searching for columns delimited by tabs (two or more consecutive spaces).

If the **WINDOW** clause is not included in the command, the current session will be scanned.

#### Example

In this example:

```
TABLE DEFINE 0 FIELDS CHAR 20 INT 3 REAL 7
RECORD FORMAT 4 (0 5) (0 25) (3 46)
SELECTION 6 16
RECORD SCAN 0
```

The name, age, and account total for each record is transferred to table 0 (zero).

---

# RECORD WRITE

---

## RECORD WRITE Table AT Record AT Position LENGTH Bytes

The RECORD WRITE command writes the contents of the record buffer to the corresponding table. DCS cannot access data contained in a table directly; it must access one record at a time through the corresponding record buffer variable, @R*n*, where *n* is a table number.Y

### Arguments

#### Table

The Table argument is a numeric specifying the table to which DCS writes the data.

#### ▼ Structured Tables

##### AT Record

The optional **AT** clause allows random access of records by setting the write position to the desired record number before performing the RECORD WRITE command.

The **Record** argument is a numeric specifying the desired write position, where the first record is record 0 (zero). If the **AT** clause is not included, the RECORD WRITE command begins writing at the end of the table, and successive writes are performed sequentially.

After DCS performs a record write, the write pointer is positioned at the next record.

#### ▼ Text Tables

##### AT Position

The optional **AT** clause allows random access of the data blocks (not text lines) by setting the write position to the specified file position before performing the RECORD WRITE command. The **Position** argument is a numeric specifying the desired write position, where the first record is record 0 (zero). If the **AT** clause is not included, the RECORD WRITE command begins writing at the end of the table, and successive writes are performed sequentially.

After DCS performs a record write, the write pointer is positioned at the next character (or line).

#### LENGTH Bytes

The optional **LENGTH** clause directs DCS to output a data block of the specified number of bytes (up to 254). The **Bytes** argument is a numeric specifying the number of bytes to be included in each data block. If the **LENGTH** clause is not included, the length of the data block written is determined by the delimiter (a carriage return character followed by a line feed character).

---

## RECORD WRITE, *continued*

---

### Example

In this example:

```
TABLE DEFINE 0 FIELDS CHAR 20
DIALOG (50, 50, 200, 75) "Edit"
EDITTEXT (20, 10, 144, 10) "Enter Name:"
BUTTON (50, 50) "OK" Resume
DIALOG END
WAIT RESUME
@R0 = EDITTEXT (1)
RECORD WRITE 0
```

a dialog, prompting for a name, is displayed. When a name is entered, the contents of the edit text field is written to table 0 (zero).



---

# REMOVE DIRECTORY

---

## REMOVE DIRECTORY Path

The REMOVE DIRECTORY command deletes the specified directory.



Note: This command does not apply to IBM TN3270 emulations.

### Arguments

#### Path

The Path argument is a string specifying the path of the directory to be removed.

### Comments

This command fails under the following conditions:

- ▼ The directory is not empty
- ▼ The path is not valid for the computer
- ▼ The path does not end with a directory name
- ▼ The directory is the current working directory.

In each case, the ERROR function returns TRUE.

### Example

This command:

```
REMOVE DIRECTORY "C:\DCSERIES\TEMP"
```

removes the directory C:\DCSERIES\TEMP.

---

# RESETSERIAL

---

## RESETSERIAL

The RESETSERIAL command resets the serial communications port of the default session window.

### Arguments

The RESETSERIAL command takes no arguments.

### Comments

On occasion, it is necessary to reset the communications port after communications transactions with some remote hosts.

The ERROR function returns TRUE if the command cannot reset the serial communications port.

### Example

In this example:

```
FILE RECEIVE BINARY "LONGFILE.TXT"  
RESETSERIAL
```

DCS resets the communications port after receiving the binary file from the remote system.

---

# RESTART

---

## RESTART

The RESTART command causes the execution of a script to branch to the first line of the executing script.

### Arguments

The RESTART command takes no arguments.

### Comments

The execution of a RESTART command does not clear any existing resources created during script execution (variables, tables, etc.).

### Example

In this example:

```
PERFORM set_up
PERFORM get_data
DIALOG
MESSAGE "Repeat Procedure?"
BUTTON "OK" RESTART
BUTTON "NO" CANCEL
DIALOG END
```

the script is restarted if the **OK** button is clicked.

---

# RESUME

---

## RESUME

The RESUME command causes script execution to resume at the command following the most recently executed WAIT command.

### Arguments

The RESUME command takes no arguments.

### Comments



Also see: **Event Handling - WAIT and WHEN Commands in Chapter 1 Introduction**

### Example

In this example:

```
DIALOG (20, 40, 200, 60) "LAUNCH"  
EDITTEXT (20, 10, 140) "Application: "  
EDITTEXT (20, 40, 140) "File Name: "  
BUTTON (50,50) "OK" PERFORM launch, RESUME  
BUTTON (110, 50) "CANCEL" CANCEL  
DIALOG END  
WAIT RESUME
```

after the **OK** button is pressed the routine labeled \*launch is performed. Then, the RESUME command branches execution beyond the WAIT RESUME command.

---

# RETURN

---

## RETURN

The RETURN command causes script execution to resume at the command following the most recently executed PERFORM command. The PERFORM and RETURN commands, when used together, define a subroutine.

### Arguments

The RETURN command takes no arguments.

### Comments

If DCS executes a RETURN command without having previously executed a PERFORM command, the RETURN command operates as the CANCEL command.

### Example

In this example:

```
IF ERROR
BEGIN
PERFORM errorRoutine ("No more data")
CANCEL
END
.
.
.
*errorRoutine ($errorMessage)
DIALOG (20, 20, 100, 60)
MESSAGE $errorMessage
BUTTON "OK" RESUME
DIALOG END
WAIT RESUME
RETURN
```

a generic error routine is created. Whenever the routine executes, the RETURN command causes execution to branch back to the line immediately following the PERFORM command that called the routine.

---

# SAVE

---

## SAVE FileName

The SAVE command saves the active session's configuration to the session file specified by the optional file name.

### Arguments

#### FileName

The optional **FileName** argument is a string specifying the session file in which to save the configuration information. The **FileName** argument must specify a valid file name for your system.

If you do not specify a file extension in the **FileName** argument, the file is automatically given the ".ses" file extension.

The **FileName** argument may also be specified as a question mark (?). DCS will then prompt you to enter the name of a file in which to save the session during script execution.

If you do not include the **FileName** argument, DCS saves the configuration to the active session file. If no session file is currently active, this command will have no effect.

### Comments

The ERROR function returns TRUE if an invalid file name is specified.

### Example

In this example:

```
SAVE "CIS"
```

the current settings are saved as the session file CIS.SES.

In this example:

```
SET EMULATION "ANSI"  
SET PHONENUMBER "5551234"  
SAVE "ANSIUSER"
```

two settings were modified and saved as session file ANSIUSER.SES.

---

# SCREEN

---

## SCREEN (x, y, w, h) Display WINDOW WinHandle

The SCREEN command controls the size and display of a session window.

### Arguments

(x, y, w, h)

The optional (x, y, w, h) coordinate set specifies the desired position and size of a session window. It indicates the top left corner (x, y), width (w), and height (h). The coordinates are specified in logical units; horizontally, there are four logical units per character; vertically, there are eight logical units per line.

### Display

The optional **Display** argument is specified by one of the following keywords:

Keyword	Description
HIDE	This keyword hides a session window.
SHOW	This keyword displays a session window if it is hidden, and updates the terminal screen with received data if it is dimmed.
DIM	This keyword turns off screen updating.
MAXIMIZE	This keyword expands a session window to fill the application window's entire client area.
RESTORE	This keyword restores a maximized session window to its previous size.

### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

### Comments

A hidden session window is fully functional.

If a window handle is not included, the command affects the active session window.

### Example

In this example:

```
SCREEN HIDE
DIAL "5551234"
PERFORM login
LOGTOFILE "INDATA.TXT"
FILE CLOSE
SCREEN SHOW
```

the session window is hidden while the incoming text is captured.

---

## SCROLL DOWN

---

### **SCROLL DOWN Rows**

The **SCROLL DOWN** command scrolls down a specified number of lines in the active session window history buffer. It is the same as clicking on the down arrow on the vertical scroll bar. It is applicable only when the vertical scroll bar is displayed (indicating that the entire host screen is not visible in the session window).

#### **Arguments**

##### **Rows**

The **Rows** argument is an integer specifying the number of lines to scroll down through the history buffer.

#### **Comments**

The result of this command is the same as if you had clicked on the down arrow on the vertical scroll bar. If the number of scroll down lines exceeds the lower limit of the history buffer, scrolling stops at the last row.

This command does not change the position of the cursor in the session window.



---

## SCROLL LEFT

---

### SCROLL LEFT Columns

The `SCROLL LEFT` command directs DCS to scroll left in the active session window. It is the same as clicking on the left arrow on the horizontal scroll bar. It is applicable only when the horizontal scroll bar is displayed (indicating the entire host screen is not visible in the session window).

#### Arguments

##### Columns

The `Columns` argument is an integer specifying the number of single character positions to scroll left.

#### Comments

The result of this command is the same as if you had clicked the left arrow on the horizontal scroll bar at the bottom of the screen. If the number of columns to scroll exceeds the left margin limit of the history buffer, scrolling stops at the leftmost column (column zero).

This command does not change the position of the cursor in the session window.

---

## SCROLL RIGHT

---

### **SCROLL RIGHT Columns**

The `SCROLL RIGHT` command directs DCS to scroll right in the active session window. It is the same as clicking on the right arrow on the horizontal scroll bar. It is applicable only when the horizontal scroll bar is displayed (indicating the entire host screen is not visible in the session window).

### **Arguments**

#### **Columns**

The `Columns` argument is an integer specifying the number of single character positions to scroll right.

### **Comments**

The result of this command is the same as if you had clicked the right arrow on the horizontal scroll bar at the bottom of the screen. If the number of columns to scroll exceeds the rightmost margin limit of the history buffer, scrolling stops at the rightmost column.

This command does not change the position of the cursor in the session window.

---

## SCROLL UP

---

### SCROLL UP Rows

The `SCROLL UP` command directs DCS to scroll up a specified number of lines in the active session window's history buffer. It is the same as clicking on the up arrow on the vertical scroll bar. It is applicable only when the vertical scroll bar is displayed (indicating the entire host screen is not visible in the session window).

### Arguments

#### Rows

The `Rows` argument is an integer specifying the number of lines to scroll up through the history buffer.

### Comments

The result of this command is the same as if you clicked the up arrow in the vertical scroll bar. If the number of lines to scroll up exceeds the upper limit of the history buffer, scrolling stops at the top row.

This command does not change the position of the cursor in the session window.

---

## SELECTION

---

### **SELECTION StartLine EndLine WINDOW WinHandle**

The **SELECTION** command selects a block of rows in a session window. Although the command selects the block, the block will not appear highlighted in the window.

#### **Arguments**

##### **StartLine**

The **StartLine** argument is a numeric specifying the number of the first row of the desired block, where the first row in a window is considered row zero.

##### **EndLine**

The **EndLine** argument is a numeric specifying the number of the last row of the desired block.

##### **WINDOW WinHandle**

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

The **WINDOW** clause selects a block in a particular session window.

#### **Comments**

The **StartLine** and **EndLine** arguments must not exceed the number of rows for a window. When you want to make a selection in a session window, the number of rows in an emulation (for example, the model in a 3270 emulation) determines the number of lines in the session window and the possible values of the **StartLine** and **EndLine** arguments. If both the **StartLine** and **EndLine** arguments are larger than the range allowed for a window, DCS will not make a selection. DCS does not make a default selection in the window.

The selection block includes all columns of the selected rows.

If you do not include the **WINDOW** clause in the command, a block in the active session window is selected.

#### **Example**

See the **RECORD SCAN** command.

---

## SELECTION APPEND

---

### SELECTION APPEND FileName TABLE WINDOW WinHandle

The SELECTION APPEND command appends the current selection in a session window to the existing contents of the specified file.

#### Arguments

##### FileName

The **FileName** argument is a string specifying the name of the file in which to save the selection. The **FileName** argument must specify a valid file name for your system or a null string (“”). If a null string is specified, a prompt appears during script execution to enter the name of a file in which to save the selection.

##### TABLE

The optional **TABLE** keyword saves the data in a tabular format. All data separated by two or more consecutive spaces will be tab delimited.

##### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

The **WINDOW** clause appends the selection from a particular session window.

#### Comments

If the **WINDOW** clause is not included in the command, the selection is appended from the active session window.

#### Example

In this example:

```
SELECTION BUFFER
SELECTION APPEND "DATAFILE.TXT"
```

DCS adds the current contents of the history buffer to the end of the file DATAFILE.TXT.

---

## SELECTION BUFFER

---

### SELECTION BUFFER WINDOW WinHandle

The SELECTION BUFFER command selects the entire contents of the session window's history buffer as a block of text. As with the SELECTION command, this command makes an invisible selection. This command is not for emulations with screens composed as pages, such as an IBM TN3270 terminal.

#### Arguments

##### WINDOW WinHandle

The optional WINDOW clause includes the WINDOW keyword and the WinHandle argument. The WinHandle argument is an integer identifying a particular session window.

#### Comments

The entire contents of the history buffer, which includes the portion displayed on the screen, is selected. If the WINDOW clause is not included in the command, the selection will be made from the active session window.

#### Example

See the SELECTION APPEND command.

---

# SELECTION PRINT

---

## SELECTION PRINT WINDOW WinHandle

The SELECTION PRINT command prints the current selection as established by the SELECTION or SELECTION BUFFER commands.

### Arguments

#### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

The **WINDOW** clause directs DCS to print the selection in a particular session window.

### Comments

If you gave the SELECTION command the proper StartLine and EndLine arguments, and if DCS has not previously executed the SELECTION command on an open window, the SELECTION PRINT command prints a blank page.

If the **WINDOW** clause is not included in the command, the selection in the active session window will be printed.

### Example

In this example:

```
SELECTION 0 23 WINDOW WINDOWHND ("SCR1")
SELECTION PRINT WINDOW WINDOWHND ("SCR1")
```

all 24 lines in the session window are sent to the printer. The WINDOWHND function is used to return the WinHandle for window SCR1.

---

# SELECTION SAVE

---

## SELECTION SAVE FileName TABLE WINDOW WinHandle

The SELECTION SAVE command saves the current selection to the specified file. This command works only with a selection made with the SELECTION or SELECTION BUFFER commands.

### Arguments

#### FileName

The **FileName** argument is a string specifying the name of the file in which to save the selection. The **FileName** argument must specify a valid file name for your system. If a null string (“”) is specified, a prompt for a file name appears.

#### TABLE

The optional **TABLE** keyword directs DCS to save the data in tabular format. All data separated by two or more consecutive spaces will be saved as tab delimited.

#### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

The **WINDOW** clause directs DCS to save a selection made in a particular session window.

### Comments

If the **WINDOW** clause is not included in the command, the selection in the current session window is saved to the file.

If a write error occurs, the Result system variable contains a string describing the error.

### Example

In this example:

```
SELECTION 0 24
SELECTION SAVE "TERMDATA.TXT"
```

DCS saves the entire contents of the session window to the file TERMDATA.TXT.



---

## SELECTION SEND

---

### SELECTION SEND WINDOW WinHandle

The SELECTION SEND command sends the selected text to the active DCS session window. This command works only with a selection made with the SELECTION or SELECTION BUFFER commands.

#### Arguments

##### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

The optional **WINDOW** clause directs DCS to send the selection in a particular session window to the remote system connected to that session.

#### Comments

If the **WINDOW** clause is not included in the command, the selection in the current window will be sent to the remote system.

#### Example

In this example:

```
SELECTION 0 0  
SELECTION SEND
```

the first line of data in the window is sent to the remote system.

---

# SEND

---

**SEND (Row, Col) NOCR String NOPROTECT ONEPACKET WAITECHO WINDOW WinHandle**

The SEND command sends text to an open DCS session window.

## Arguments

### (Row, Col)

These arguments apply only to block mode emulations. The optional **(Row, Col)** coordinate set indicates the row and column of where to place the text in a window. The first row in the window is row 9 (zero), and the first column is column 0 (zero). If coordinates are not included in the command, text is placed in the window starting at the current cursor position. These coordinates are especially useful when connecting to remote systems requiring specialized or complex interactions.

### NOCR

A carriage return is automatically sent following the contents of the **String** argument. The optional **NOCR** keyword directs DCS not to append a carriage return to the **String** argument.



**Note:** For **IBM TN3270 emulations**, the SEND command automatically appends {NEWLN} to the end of a line in place of a carriage return. Use **NOCR** to remove the {NEWLN}.

### String

The **String** argument contains the text to send to a window.



Also see: Rules for Sending Special Strings at the end of this command description.

### NOPROTECT

The optional **NOPROTECT** keyword ignores protected areas of the screen and inserts text to the screen buffer. This does not require the host to read back the modified protected fields. This string should only be used in special situations.



**Note:** This string is applicable only for IBM 3270/5250 emulations.

### ONEPACKET

The optional **ONEPACKET** keyword requires DCS to send the contents of the **String** argument as a single network packet.



**Note:** For IBM TN3270 emulations, the optional **ONEPACKET** keyword does not apply.

---

## SEND, *continued*

---

### WAITECHO

The optional **WAITECHO** keyword sends the string argument one character at a time, waiting until the remote system echoes back each character before sending the next character in the string. This process continues until all characters in the **String** argument are sent.



**Note:** For IBM TN3270 emulations, the optional **WAITECHO** keyword does not apply.

### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

The **WINDOW** clause sends the string to the window of a particular remote system.

### Comments

You must transmit a null character as follows:

```
Send NoCR ``^@``
```

This script example transmits a null character without a carriage return. The string `^@` represents a null character.

If a null character is embedded in a string, the null character is sent with the other characters in the string. For example, consider the following sequence of script commands:

```
$String = "abcd^@efgh"  
Send $String
```

The **SEND** command sends the text to the active window, just as though you had typed the text string from the keyboard. The **SEND** command transmits 1) the letters a, b, c, and d, 2) then, the null character, and 3) then, the letters e, f, g, and h.

If the **WINDOW** clause is not included, the text is sent to the active session window.

---

## SEND, *continued*

---

### Example

In this example:

```
SEND CHR (27) | CHR (43)
```

the first CHR function returns the escape character, and the second CHR function returns the plus character. The Escape and Plus characters are concatenated (joined) together into a string with the concatenation operator (the vertical bar or pipe symbol). The SEND command then sends the string, followed by a carriage return character, to a remote system.

In this example:

```
SEND "{insertoff}18934
```

the {insertoff} metakey ensures that the input mode is Overtyping before sending the character string. After sending the character string, the input mode returns to its previous setting.

In this example:

```
DIAL "5551234"  
SEND NOCR "^C"  
WAIT STRING "Enter password: "  
SEND "secretcode"
```

the specified phone number is dialed, a [Ctrl]+[C] without a carriage return is sent, and then the script waits for the string "Enter password:" to appear in the session window. When the string appears, the string "secretcode" followed by a carriage return is sent.

In this example:

```
CONNECT  
SEND NOCR "{ENTER}{WAITXC}{ENTER}"  
WAIT STRING "Enter password: "  
SEND NOCR "secretcode {ENTER}"
```

a connection to a remote IBM 3270 system is made, an {ENTER} key is sent, and the script waits for the XClock to disappear at which occurrence another {ENTER} key is sent.

The string {WAITXC} is a special string in DCS. {WAITXC} in a string indicates transmission with a remote system. DCS waits for the remote system to become ready again before sending the remaining characters. Note that **NOCR** suppresses the automatic {NEWLN} normally appended to any line transmitted with the SEND command.

After the SEND command, the script waits for the string "Enter password: " to appear in the currently active session window. When it appears, the string "secretcode" is sent, followed by an {ENTER} key.

---

## SEND, *continued*

---

In this example:

```
CONNECT "SCR1" %hwnd
SEND NOCR (20,10) "userid{ENTER}" WINDOW %hwnd
WAIT STRING "Enter Password: "
SEND NOCR "password{ENTER}" WINDOW %hwnd
```

A connection to host session SCR1 is made. The user's ID is sent to the host starting from at 20, column 10. After the remote system prompts for a password, DCS sends the password with a carriage return starting from the current cursor position.

When referencing a row or column, a variable or constant must be used, i.e., an expression cannot be used. For example:

```
%row=1
SEND (1,%col)
SEND (1,%col+1)
SEND (1,(%col+1))
```

is not supported. The example above could be re-written as:

```
%row=1
SEND (1,%row);column 1 row 1
%row=%row+1 or INCREMENT %row
SEND (1,%row);column 1 row 2
%row=%row+1 or INCREMENT %row
SEND (1,%row);column 1 row 3
%row=%row+1 or INCREMENT %row
```

In this example:

```
SEND NOCR $UserID | "{TAB}" | $Password | \ "{ENTER}"
{WAITXC}PROFS{ENTER}"
```

DCS sends a user's ID, a [TAB] character, a user's password, an [ENTER] character, and then waits for the remote system to process that information before sending the PROFS command and an [ENTER] character to the remote system.

In this example:

```
SEND (2,12) NOCR "text" NOPROTECT
```

A text string is inserted in the screen buffer at position 2,12, even if this is a protected area.

---


## SEND, *continued*

---

### Rules for Sending Special Strings

Use special strings enclosed in braces { } or preceded by a caret (^) symbol to send specific keys or perform actions. The following table summarizes the available special strings or actions:

Key or Action	String
{	"{"
^	"^"
launch a script	'^\$Escript.dct ' <i>or</i> {^\$Escript.dct} <i>or</i> '^\$Pscript.dct ' <i>or</i> {^\$Pscript.dct} <i>or</i>
launch an application	'^\$Xapplication.exe parameters ' <i>or</i> {^\$Xapplication.exe parameters}
set the result string	'^\$Rresult string ' <i>or</i> {^\$Rresult string}  Setting the result string via a SEND command affects the result string only for that script. It does not affect any other script.
specific key	"{key label}"  To obtain the key label for a specific key, open the <b>Session Properties</b> dialog. Select an emulation and then select the <b>Keyboard</b> tab. Click the <b>Edit</b> button and then select the <b>Terminal</b> tab in the mapping options area. Any string listed in the <b>Terminal</b> tab may be enclosed within braces { } and sent as a meta key.  The key label you enclose in braces is not case sensitive, but internal spaces must be preserved.
turn off Insert mode	"{insertoff}character string"
wait for host ready state	Use the special string {WAITXC}  DCS waits for the remote system to become ready again before it sends the remainder of the characters.

 Note: Applies only to Block Modes for IBM TN3270 and Tandem 6530 emulations.

---

---

# SENDBREAK

---

## SENDBREAK DelayUnits

The SENDBREAK command sends the break signal for a specified number of milliseconds.



Note: This command does not apply to IBM TN3270 emulations.

### Arguments

#### DelayUnits

The DelayUnits argument is a numeric specifying the length the break is to be sent, where one delay unit equals one millisecond.

### Example

In this example:

```
SEND "password"  
SENDBREAK 30
```

a 30 millisecond break is sent to the remote system after sending the password.

---

# SET

---

## SET Variable Source

The SET command assigns the value of the **Source** argument to the specified variable.

### Arguments

#### Variable

The **Variable** argument specifies the name and type of the variable (either a string variable, numeric variable, or Boolean variable) to be assigned.

#### Source

The **Source** argument specifies the expression whose contents are assigned to the variable. The Source argument must be of the same type (string, numeric, or Boolean) as the specified variable.

### Comments

The SET command is equivalent to the assignment (=) operator.

### Example

In this example:

```
SET $name "Arnold Wilson"
```

the string variable \$name is assigned the string Arnold Wilson.

These two examples:

```
SET %yearprofits (%income - %expenses) * 12
```

```
%yearprofits = (%income - %expenses) * 12
```

are equivalent. The numeric variable %yearprofits is assigned the value of the complex numeric (%income - %expenses) multiplied by 12.



---

# SET APPTITLE

---

## SET APPTITLE Name

The SET APPTITLE command sets the title caption bar of the application window.



**Note:** For scripts that produce multiple session windows in a single execution of the DCS application rather than multiple application windows, use the SET WINDOWTITLE command. Otherwise, the session window title is lost when session windows are re-sized.

## Arguments

### Name

The **Name** argument is a string specifying the text to display in the title bar of the application window.

## Comments

Unlike the TITLE command, which is a compiler directive and requires a literal string (characters within quotation marks, not a string variable), the SET APPTITLE command is a true script command and can accept a string expression of any format.

As with the TITLE command, the effects of the SET APPTITLE command end when the script terminates. When the script ends, the title of the application window reverts to the default.

## Example

In this example:

```
SET APPTITLE "Mail System for " | $username
```

the SET APPTITLE command overrides the default title and displays the character sequence "Mail System Acme Corporation" in the application window title bar. (The default title of the application window is DCS.)

---

# SET ATTRIBUTES

---

## SET ATTRIBUTES FileName Attributes

The SET ATTRIBUTES command modifies the file attributes of the specified file.

### Arguments

#### FileName

The FileName argument is a string specifying the name of the file. The FileName argument must specify a valid file name for your system.

#### Attributes

The Attributes argument is a numeric specifying the file attributes as shown on the table below. You may specify the Attributes argument as a sum of the values of the attributes listed below. Specifying an Attributes argument of 0 (zero) removes all attributes from the file.

Value	File Attribute
1	Read-only File
2	Hidden File
4	System File
16	Subdirectory
32	Archived File
64	Compressed File

### Example

In this example:

```
IF ATTRIBUTES ($file) = 3
SET ATTRIBUTES $file 6
```

In this example, if the attributes of \$file are read-only and hidden (the sum of their values is 3 (three)), DCS sets the attributes to Hidden and System (the sum of their values is 6 (six)).

---

# SET AUTOSCROLLTOCURSOR

---

## SET AUTOSCROLLTOCURSOR Boolean

The SET AUTOSCROLLTOCURSOR command determines whether to scroll the session window to the emulation cursor position each time the cursor moves outside of the visible part of the session window.

### Arguments

#### Boolean

The Boolean argument is a Boolean value specifying whether or not the **Autoscroll** option is enabled. If the Boolean argument evaluates to TRUE, scrolling occurs automatically when the emulation cursor is moved outside the visible part of the session window. If the Boolean argument evaluates to FALSE, automatic scrolling of the session window is turned off.

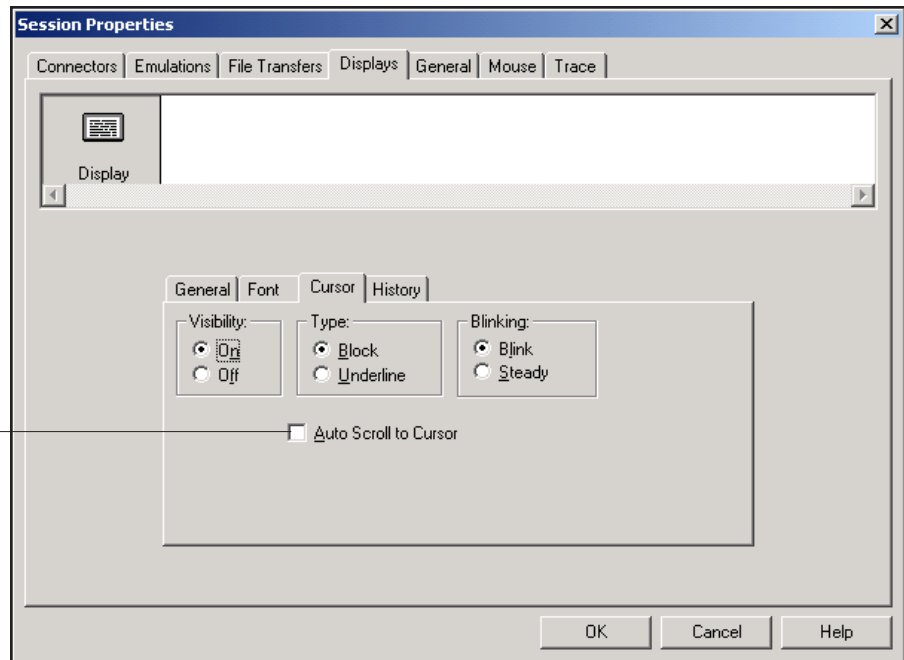
### Comments

Vertical or horizontal scrolling is applicable only when the appropriate scroll bars are visible in the session window.

This command has the same effect as checking or unchecking the **Autoscroll to Cursor** option on the **Cursor** subtab of the **Displays** tab of the **Session Properties** dialog.

**Figure 3.1**  
Cursor sub-tab of  
the Displays tab in  
the Session Proper-  
ties dialog

Auto Scroll to  
Cursor option



Where possible use the DISPLAYCONFIG command instead of this command.

### Example

```
SET AUTOSCROLLTOCURSOR TRUE
```


---

# SET AUTOSIZE

---

## SET AUTOSIZE Boolean

The SET AUTOSIZE command has the same effect as checking or unchecking the **Autosize Font to fit session window** check box on the **Displays** tab of the **Session Properties** dialog.

 Note: This command does not apply to IBM TN3270 emulations.

### Arguments

#### Boolean

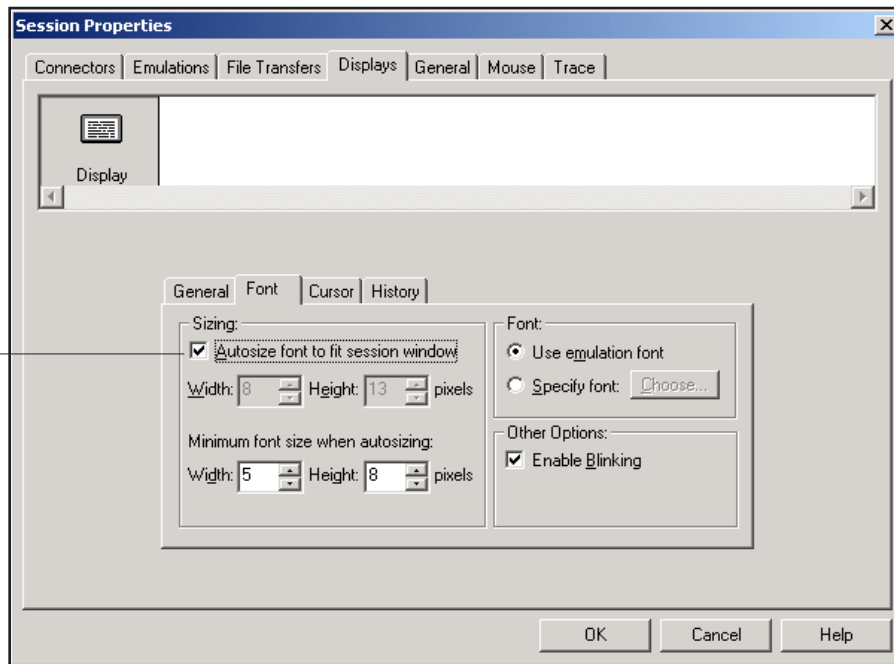
The Boolean argument is a Boolean expression. If the expression evaluates to TRUE, **Autosize Font** is enabled. If the expression evaluates to FALSE, **Autosize Font** is disabled.

### Comments

Where possible use the DISPLAYCONFIG command instead of this command.

Figure 3.2  
Font sub-tab of the  
Displays tab in the  
Session Properties  
dialog

Autosize font to fit  
session window




---

# SET BACKSPACEDESTRUCTIVE

---

## SET BACKSPACEDESTRUCTIVE Boolean

The SET BACKSPACEDESTRUCTIVE command causes the [BACKSPACE] key to erase, or delete, characters as the cursor moves from right to left.

 **Note:** This command does not apply to IBM TN3270 emulations.

### Arguments

#### Boolean

The Boolean argument is a Boolean expression. If the expression evaluates to TRUE, the [BACKSPACE] key erases, or deletes, characters as the cursor moves from right to left. If the expression evaluates to FALSE, the cursor moves over characters without deleting them when the [BACKSPACE] key is pressed.

### Example

In this example:

```
If #HostSupportsDestructBack
SET BackSpaceDestructive OFF
```

DCS checks to see if the remote system supports destructive backspaces (by checking to see if the Boolean variable #HostSupportsDestructBack evaluates to TRUE). If the remote system does support a destructive backspace, DCS uses the Boolean value OFF ( FALSE) to set the [BACKSPACE] key to a non-destructive mode, allowing the [BACKSPACE] key to move the cursor backward over characters without deleting them.

---

# SET BACKSPACEKEY

---

## SET BACKSPACEKEY Keyword

The SET BACKSPACEKEY command determines whether the cursor moves from right to left sent when the backspace key is pressed, or instead deletes the character at the current cursor location.



Note: This command does not apply to IBM TN3270 emulations.

### Arguments

#### Keyword

The Keyword argument is specified by one of the following keywords:

Keyword	Action
BACKSPACE	Directs DCS to send a backspace character when the backspace key is pressed.
DELETE	Directs DCS to send a delete character when the backspace key is pressed.

### Example

In this example:

```
LOAD "MAILSET"  
SET BACKSPACEKEY DELETE  
PERFORM gather_data  
SET BACKSPACEKEY BACKSPACE
```

the character sent by pressing the [BACKSPACE] key is changed to a delete character only while the `gather_data` routine is performed.

---

# SET BAUDRATE

---

## SET BAUDRATE Rate

The SET BAUDRATE command changes the baud rate of the selected communications port.



**Note:** This command does not apply to IBM TN3270 emulations.

### Arguments

#### Rate

The **Rate** argument is a numeric specifying the desired baud rate.

The following baud rates are available :

110  
300  
600  
1200  
2400  
4800  
9600  
14400  
19200  
28800  
38400  
57600  
115200

### Example

In this example:

```
SET BAUDRATE 19200  
DIAL "5551234"
```

the baud rate is set to 19200 before dialing the phone number.

---

# SET BINARYTRANSFERPARAMS

---

## SET BINARYTRANSFERPARAMS OptionKeyWord ActionKeyWord OptionKeyWord ActionKeyWord...

The SET BINARYTRANSFERPARAMS command sets the parameters for the active session's binary file transfer protocol.



**Note:** This command does not apply to IBM TN3270 emulations.

### Arguments

#### OptionKeyWord ActionKeyWord

The OptionKeyWord and ActionKeyWord arguments form a clause. OptionKeyWord relates to the name of a parameter in a settings dialog for a binary transfer protocol. The ActionKeyWord defines the setting for a parameter indicated by the OptionKeyWord.



**Note:** Though all of the following keywords are optional, the SET BINARYTRANSFERPARAMS command requires at least one or more of the following keyword clauses (in any order):

Keyword	Action
ZAUTOSTART ZmodemAutoStart	When DCS is applying the ZModem protocol, the optional ZAUTOSTART clause indicates whether to transfer a file when DCS receives a ZModem download auto-start sequence from the remote system. The ZmodemAutostart argument should evaluate to a Boolean expression or keyword. The Boolean keywords ON, YES, and TRUE evaluate to a true Boolean value. If ZmodemAutostart evaluates to TRUE, DCS transfers a file automatically when it receives a ZModem download auto-start sequence. The Boolean keywords OFF, NO, and FALSE evaluate to a false Boolean value. If ZmodemAutostart evaluates to FALSE, DCS will transfer a file only when you activate the Send File option in the Session menu or a script contains a FILE SEND BINARY command. This keyword clause has the opposite effect of the Disable AutoStart check box in the ZModem dialog (found in the File Transfers tab of the Session Properties dialog).
ZERRORCHECK ZmodemErrorCheck	The optional ZERRORCHECK clause sets the error checking method for ZModem transfers. The ZmodemErrorCheck argument should be one of the following keywords: CRC32, or CRC16. This keyword clause has the same effect as selecting a radio button in the Error Checking group box of the ZModem Settings dialog box.
ZEXISTING ZmodemFileExists	The optional ZEXISTING clause tells DCS how to handle a file on a receiving system, when the file name of a file on the receiving system matches that of the file DCS is transferring. The ZmodemFileExists argument should be one of the following keywords: RESUME, PROMPT, OVERWRITE, SKIP. This keyword clause has the same effect as selecting a radio button in the If File Exists group box of the ZModem Settings dialog box.



---

## SET BINARYTRANSFERPARAMS, *continued*

---

KEYWORD	ACTION
ZTIMING ZmodemTiming	The optional ZTIMING clause sets the time-out options for ZModem transfers. The ZmodemTiming argument should be one of the following keywords: STANDARD, or LOOSE. This keyword clause has the same effect as selecting a radio button in the Timing Constraints group box of the ZModem Settings dialog box.

### Comments

Wherever possible, use the XFERCONFIG command instead of this command.

### Example

This example:

```
SET BINARYTRANSFERPARAMS ZEXISTING OVERWRITE
```

overwrites a file that already exists on a receiving system, when transferring files via ZModem.

---

## SET BINARYTRANSFERS

---

### SET BINARYTRANSFERS Protocol WINDOW WinHandle

The SET BINARYTRANSFERS command sets the binary file transfer protocol or host environment for a session.

#### Arguments

##### Protocol

The Protocol argument is a keyword specifying the binary transfer protocol.

The Protocol argument is specified by one of the following keywords:

- IND\$File
- IXF
- XMODEM
- YMODEM
- ZMODEM
- KERMIT

##### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument.

The **WinHandle** argument is an integer identifying a particular session window.

#### Comments

If the **WinHandle** argument is not included, the transfer protocol is applied to the active session window.

To configure the parameters for the specified file transfer protocol, use the XFERCONFIG command.

#### Example

In this example:

```
SET BINARYTRANSFERS ZMODEM
FILE SEND BINARY "SESCRIPT.DCT"
```

ZModem is established as the file transfer protocol before performing a file transfer.

---

# SET BUFFERLINES

---

## SET BUFFERLINES Lines

The SET BUFFERLINES command specifies the total number of 80-column lines to be reserved for the session window and history buffer. It only applies to emulations that are not in block mode.

### Arguments

#### Lines

The **Lines** argument is an integer specifying the number of lines to reserve. DCS always displays at least 25 lines in the session window. If a number less than 25 is specified, a default value of 25 is used.

### Comments

The default history buffer size is 100 lines.

If there is not enough memory to allocate a buffer of the desired size, DCS includes as many lines as possible by allocating available memory to the buffer. You may increase the buffer size to a maximum of 9,999 lines.

This command has the same effect as selecting the number of history buffer lines in the History tab in the Displays section of the Session Properties dialog.

Wherever possible, use the DISPLAYCONFIG command instead of this command.

### Example

In this example:

```
SET BUFFERLINES 200
```

a history buffer consisting of 200 lines is established.

---

# SET CARRIERDETECT

---

## SET CARRIERDETECT Boolean

The SET CARRIERDETECT command sets the carrier detect flag.



**Note:** This command does not apply to IBM TN3270 emulations.

### Arguments

#### Boolean

The Boolean argument is specified by a Boolean expression or keyword. The Boolean keywords ON, YES, and TRUE evaluate to TRUE. The Boolean keywords OFF, NO, and FALSE evaluate to FALSE.

If the Boolean evaluates to TRUE, DCS uses the modem's carrier status to determine if the carrier is present. If the Boolean evaluates to FALSE, DCS logically determines if the carrier is present.

### Comments

DCS will automatically detect carrier loss only if the carrier detect flag is TRUE. If the carrier is lost and the carrier detect flag is FALSE, DCS must hang up before dialing again.

### Example

In this example:

```
SET MODEM HAYES
SET CARRIERDETECT TRUE
DIAL "5551234"
```

DCS establishes the modem type and the carrier detect flag before attempting to dial.

---

# SET COLUMNS

---

## SET COLUMNS Integer

The SET COLUMNS command specifies the number of columns in the session window, where there is one character per column.

### Arguments

#### Integer

The Integer argument must be a valid column width value for the emulation. The value is typically the integer 80 or 132.

### Comments

This command applies to all emulations that support variable column width.

### Example

In this example:

```
SET COLUMNS 132
PERFORM display_data
SET COLUMNS 80
```

DCS switches to 132-column mode while the `display_data` routine is performed. The standard 80-column mode is then restored.


---

# SET CONNECTION

---

## SET CONNECTION Connector Command WINDOW WinHandle

This command allows you to specify which type of communications connector DCS uses when it connects to a remote system. This command can also load a DLL created by a third party. For further information about third party DLLs and about creating a communications connector for DCS, contact FutureSoft.

 **Note:** This command does not apply to IBM TN3270 emulations.

### Arguments

#### Connector

The **Connector** argument is a string specifying the name of the communications connector. The argument can also specify a DLL containing a valid DCS connector. Connector DLLs for DCS are located in the Conn directory by default. Valid strings are shown in the following table:

Connector	String
Direct Serial	COMDIR
Modem	COMTAPI or TAPI
Meridian LAT32	MERDNLAT
Telnet	WINSOCK or TELNET
Microsoft SNA Server*	FMIDLL*

\* available only in appropriate Client Option package

#### Command

The optional **Command** argument is a string specifying any optional commands that should be passed to the communications connector during its initialization. These commands are specific to the communications connector and its DLL.

Each command string consists of a phrase or a number of phrases separated by a unique separator character:

#### CommandPhrase Separator CommandPhrase Separator...

In the command string description, **Separator** is a single character, which is not alphanumeric, and does not appear in any **CommandPhrase** specified for a connector. For example, the address of a node on a TELNET network commonly includes numbers with periods to separate parts of the address. The address could be a **CommandPhrase**, but since a period is part of the address, you must use a character other than a period (possibly a semicolon) to separate each **CommandPhrase** from other **CommandPhrases** in the command string. A **Separator** must follow each **CommandPhrase**. If you omit a phrase, DCS uses a default value for that setting, but the **Separator** for that **CommandPhrase** must still appear in the command string. For example, if a connector needed three **CommandPhrases**, but you want to use a

---

## SET CONNECTION, *continued*

---

default or previous setting for the second **CommandPhrase**, the **Command** argument might appear as follows:

In this example:

```
“CommandPhrase1;;ComandPhrase3;”
```

a semicolon is the **Separator**.

The command string, like all strings in DCS may have a maximum of 254 characters. If you exceed this limit, the results are unpredictable.

Below is a list of valid command phrases and keywords.

CommandPhrase	Action
COM <i>n</i>	The COM clause directs DCS to use a serial communications port as the communications connector. The <i>n</i> argument is an integer which specifies a communications port. The value of <i>n</i> may be 1, 2, 3, 4, 5, 6, 7, 8, or 9. If the port is not valid for your computer, DCS displays the message “COM Port <i>n</i> Not Available” in a dialog box.
WINSOCK “HostAddr Separator Port Separator”	The HostAddr argument indicates either a host name located in your Hosts file or the Internet protocol address of the remote system.  The Port argument is an integer specifying the TCP/IP port to which you wish to connect. Unless you are connecting to a specialized device, the port should be set to 23.

---

### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular child window.

If a window handle is not specified, the command affects the active session window.

### Comments

Wherever possible use the **CONNCONFIG** command (rather than the optional **Command** argument) to configure the connector you specify with this command.

### Example

In this example:

```
SET CONNECTION “WINSOCK” “MYHOST;23;”
```

the connector is set to WINSOCK and connects to the remote system named “MYHOST” using port 23.

---

## SET CONNECTMESSAGE

---

### SET CONNECTMESSAGE Text

The SET CONNECTMESSAGE command changes the characters in the message contained in the ConnectMessage system variable.



**Note:** This command does not apply to IBM TN3270 emulations.

### Arguments

#### Text

The Text argument is the string displayed when a connection is made to a remote system.

### Comments

Compare with the CONNECTMESSAGE function.

### Example

In this example:

```
$NewMessage = "SUCCESS"  
SET ConnectMessage $NewMessage
```

DCS places the text Success in the \$NewMessage string variable. The string variable becomes the argument for the SET CONNECTMESSAGE command.



---

# SET CONNECTRESULT

---

## SET CONNECTRESULT Num

The SET CONNECTRESULT command changes the value of the ConnectResult system variable.



**Note:** This command does not apply to IBM TN3270 emulations.

### Arguments

#### Num

The Num argument is an integer which the command places into the ConnectResult system variable. The integer, which must be one of the values listed in the table below, corresponds to a specific modem state.

Value	Modem State
0	Success
3	No Carrier
4	Modem Error
6	No Dial Tone
7	Busy
8	No Answer
99	User-defined Error

### Example

In this example:

```
Dial
SWITCH ConnectResult ( )
Case 6:
SET ConnectResult 99
PERFORM "EndScript"
Case 7:
PERFORM "CallAgain"
LEAVE
Switch End
```

DCS checks the ConnectResult system variable after dialing. If the value of the variable is six (no dial tone), DCS resets ConnectResult to a user-defined error and ends the script; however, if the value of the variable is seven (the line was busy), DCS calls the number again.

---

# SET CURSOR

---

## SET CURSOR Display

The SET CURSOR command sets the shape and state of the cursor in a session window.

### Arguments

#### Display

The Display argument is specified by one of the following keywords:

Keyword	Action
ON	Turns on the cursor display
OFF	Turns off the cursor display
BLOCK	Sets the cursor shape to a block
UNDERLINE	Sets the cursor shape to an underscore character

### Comments

If the cursor is set to OFF, setting the shape to BLOCK or UNDERLINE changes the cursor character, but does not turn the display to ON. This must be done with a separate SET CURSOR ON command.

Wherever possible use the DISPLAYCONFIG command instead of this command.

### Example

In this example:

```
SET CURSOR BLOCK
```

DCS changes the shape of the cursor to a block.

---

## SET DATABITS

---

### SET DATABITS Integer

The SET DATABITS command sets the number of data bits DCS uses during serial transmission.



**Note:** This command does not apply to IBM TN3270 emulations.

### Arguments

#### Integer

The **Integer** argument specifies the desired number of data bits. The **Integer** argument may be one of the following values:

4      5      6      7      8

### Example

In this example:

```
SET DATABITS 7
;connect to a remote system
SET PARITY EVEN
;configure the remote system
SET BINARYTRANSFERS KERMIT
;to receive a file via the kermit transfer protocol
FILE SEND BINARY "UTILITY.DCT"
```

communications settings are established before sending the file `UTILITY.DCT`.

---

## SET DDETIMEOUT

---

### SET DDETIMEOUT Seconds

The SET DDETIMEOUT command sets the timeout period for the following DDE commands: Instruct, Request, Table Reply, Poke, Table Poke

#### Arguments

##### Seconds

The optional **Seconds** argument is an integer identifying the time period to wait for the above commands to complete. The default value is 20 seconds.

#### Comments

The ERROR function is set to TRUE if the DDE command times out.

---

## SET DECIMAL

---

### SET DECIMAL Numeric

The SET DECIMAL command specifies the number of places for string representations of real numerics.

#### Arguments

##### Numeric

The Numeric argument specifies the desired number of decimal places.

#### Comments

This command does not affect the internal precision of the real numerics, only the format.

Use the DECIMAL command in conjunction with the STR function.

#### Example

In this example:

```
SET DECIMAL 2
DISPLAY STR (1000*.0875)
```

the string 87.50 is displayed in the session window.

---

# SET DEFAULTSESSIONHANDLE

---

## SET DEFAULTSESSIONHANDLE WinHandle

The SET DEFAULTSESSIONHANDLE command assigns an integer to the DefaultSessionHandle system variable.

### Arguments

#### WinHandle

The WinHandle argument is an integer expression specifying the window handle of the default session window.

### Comments

In many cases, the default session window is the active session window. With multiple sessions, however, you may wish to specify a default session window as the target of those commands which do not target a specific session window (rather than having those commands target the active session window).

### Example

In this example:

```
%SessHandle = WINDOWHND ("VT220")
SET DEFAULTSESSIONHANDLE %SessHandle
```

the default session window handle for this script is set to the handle of the window whose title is VT220. In this example, VT220 may or may not be the currently active session. Regardless, it is now the default session for any command which does not target a specific session window.

---

# SET DIRECTORY

---

## SET DIRECTORY Type CREATE Path

The SET DIRECTORY command specifies where to maintain the file directory for the specified file type.

### Arguments

#### Type

The **Type** argument specifies the file type. It is specified by one of the following keywords:

DNLOAD    MAPS    MEMO    SETTINGS    SCRIPT    TASK    UPLOAD

#### CREATE

The optional **CREATE** keyword directs DCS to create the specified directory if it does not exist.

#### Path

The **Path** argument is a string specifying the path in which to maintain the indicated type of directory. The **Path** argument must specify a valid directory for your system.

### Comments

If the **CREATE** keyword is not included, the **ERROR** function returns TRUE if the specified directory does not exist. If the **CREATE** keyword is included, the **ERROR** function returns TRUE if the specified directory cannot be created or if it already exists. If the directory already exists or if the subdirectories leading to the directory do not exist, the **ERROR** function returns TRUE.

You can use the SET DIRECTORY command to create a directory that doesn't exist by including the **CREATE** keyword. This allows you to create a directory and make it the default directory with a single command.

### Example

In this example:

```
SET DIRECTORY TASK CREATE "C:\DCSERIES\TASK"
```

DCS is directed to maintain all executable scripts in the DCSERIES\TASK directory on drive C:. If this directory does not exist, DCS creates it.

In this example:

```
$TASKDIR = "C:\DCSERIES\TASK"  
CREATE DIRECTORY $TASKDIR  
SET DIRECTORY TASK $TASKDIR
```

creates the directory C:\DCSERIES\TASK and then sets the directory as the default storage location for script task files.

---

# SET EMULATION

---

## SET EMULATION Emulation WINDOW WinHandle

The SET EMULATION command selects a terminal emulation.



**Note:** This command does not apply to IBM TN3270 emulations.

### Arguments

#### Emulation

The Emulation argument is a string specifying the name of the emulation. The argument can also specify a DLL containing a valid DCS emulation. Emulation DLLs for DCS are located in the Emul directory by default. The emulations found in this directory include those which are installed. Valid strings for the Emulation argument are shown in the following table:

Base Product Emulations	
Emulation	String
ADDS Viewpoint/60	ADDSVP60
ANSI/TTY*	ANSI
AT&T 605/705*	ATT605
VT Series*	VT420
Teletype*	TV950
Wyse 50/60	WYSE

Client Option Emulations	
Emulation	String
AT&T 4425	ATT4425
HP 70092/94	HP70094
Tandem 6530	Tandem
TN3270	IBM3270
TN3287 (print session)*	IBM3287
TN5250	IBM5250

\* Use the EMULCONFIG command to specify exact model.

The emulation is loaded with default settings; none of the settings in the previous emulation are transferred to the new emulation.

#### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

If a window handle is not specified, the command affects the active session window.



---

## SET EMULATION, *continued*

---

### Comments

If the session whose emulation type you wish to set was loaded via script, a default session handle must be specified before using the SET EMULATION command, or the session's window handle must be specified.

### Example

In this example:

```
SET EMULATION "VT420"
```

the DEC VT420 emulation is loaded for the current session.

In this example:

```
LOAD
SET DEFAULTSESSIONHANDLE (ACTIVE())
SET EMULATION "VT420"
EMULCONFIG "MODEL=vt220"
```

the script loads a new session. The SET DEFAULTSESSIONHANDLE specifies which session will be the target of the SET EMULATION command. The ACTIVE() function is used to obtain the window handle of the new session. The emulation is set to VT Series, and the EMULCONFIG command is used to specify the VT220 model type.

---

## SET FKEYSSHOW

---

### SET FKEYSSHOW Boolean

The SET FKEYSSHOW command automatically displays the session toolbar buttons when you load a session properties file.

#### Arguments

##### Boolean

The Boolean argument is specified by a Boolean expression or keyword. The Boolean keywords ON, YES, and TRUE evaluate to TRUE. The Boolean keywords OFF, NO, and FALSE evaluate to FALSE.

#### Comments

By default, any emulation which supports the use of session function keys will automatically display the keys (F1 through F8) at the bottom of the session window.

If the Boolean evaluates to TRUE, DCS automatically displays the session toolbar buttons.

#### Example

In this example:

```
set fkeysshow off
load "session.ses"
```

when the session file is loaded, the session function keys are not automatically displayed.

---

# SET FLOWCONTROL

---

## SET FLOWCONTROL Type

The SET FLOWCONTROL command specifies the desired flow control method.



**Note:** This command does not apply to IBM TN3270 emulations.

## Arguments

### Type

The **Type** argument is specified by one of the following keywords:

HARDWARE XONXOFF NONE

## Comments

HARDWARE flow control is equivalent to the RTS/CTS flow control.

## Example

In this example:

```
SET FLOWCONTROL XONXOFF
PERFORM send_data
```

the flow control is set to XON/XOFF before performing the routine `send_data`.

---

# SET KEEPPRINTCHANNELOPEN

---

## SET KEEPPRINTCHANNELOPEN KBoolean

The SET KEEPPRINTCHANNELOPEN command specifies whether DCS will relinquish the Windows print channel when the remote system closes an active print job.

### Arguments

#### KBoolean

The KBoolean argument is a Boolean expression or keyword. The Boolean keywords ON, YES, and TRUE evaluate to TRUE. The Boolean keywords OFF, NO, and FALSE evaluate to FALSE.

If the Boolean evaluates to TRUE, DCS keeps the Windows print channel open even after the remote system has closed its active print job. If the Boolean evaluates to FALSE, DCS relinquishes the Windows print channel whenever the remote system closes its print job.

### Comments

The Windows printer is a shared resource. Usually, Windows applications open the printer resource, use it, and then relinquish it to Windows. Between the closing and opening of the printer resource, Windows issues a form feed to clear out any residual printing from the previous print job.

When a remote system sends printer commands to DCS, it may repeatedly open and close the print channel. In this case, if DCS were to open and close the Windows printer resource every time the remote system opened or closed the print channel, you would waste paper.

The SET KEEPPRINTCHANNELOPEN command makes it possible for separate printing requests from a remote system to appear as one printing request to Windows. However, if you have more than one session window, and if the remote system in each session is printing, DCS considers the print requests from each session as separate Windows printing tasks; therefore, you can keep the printing channel open or closed on a session per session basis: keep all open, keep all closed, or keep some open and others closed.

### Example

This example:

```
SET EMULATION "VT420.DLL"  
EMULCONFIG "MODE=VT220"  
SET KEEPPRINTCHANNELOPEN ON  
WHEN STRING 1 "Starting Print" DISPLAY "^[[5i"  
WHEN STRING 2 "Stopping Print" DISPLAY "^[[4i"  
WHEN STRING 3 "Closing Print" PRINT CLOSE  
WAIT RESUME
```

loads the DEC VT-220 emulation and opens the print channel. The script then sets up three WHEN STRING commands to watch for incoming data from the remote system.

When the data enters the session window, the command block of the WHEN STRING commands sends printing control commands to the emulation and to DCS.

---

## SET KEEPPRINTCHANNELOPEN, *continued*

---

When DCS receives the string `Starting Print` from the remote system, the script sends the string `^[5i` (hexadecimal, 1B 5B 35 69) to the session window and to the emulation. This character sequence is the VT-220 escape sequence which requires the emulation to send data it is receiving from the remote system to a printer.

When DCS receives the string `Stopping Print`, the script sends the string `^[4i` (hexadecimal, 1B 5B 34 69) to the session window and to the emulation. This VT-220 escape sequence stops the emulation from sending data to the printer.

When DCS receives the string `Closing Print`, the script closes the Windows printer resource.

---

# SET LOCALECHO

---

## SET LOCALECHO Boolean

The SET LOCALECHO command controls the display of local keystrokes.



**Note:** This command does not apply to IBM TN3270 emulations.

### Arguments

#### Boolean

The Boolean argument is specified by a Boolean expression or keyword. The Boolean keywords ON, YES, and TRUE evaluate to TRUE. The Boolean keywords OFF, NO, and FALSE evaluate to FALSE.

If the Boolean evaluates to TRUE, DCS displays keystrokes in the local session window as well as sending them to the remote system. If the Boolean evaluates to FALSE, DCS sends keystrokes to the remote system only.

### Comments

The effect of this command depends on the terminal emulation you have selected for a session. If the emulation allows you to turn on and off local echo, this command will affect that setting.

### Example

In this example:

```
DIAL $number
IF CONNECT ( )
SET LOCALECHO TRUE
```

local echo is turned on if a connection is made.

---

# SET NETID

---

## SET NETID Text

The SET NETID command changes the contents of the NetID system variable.



**Note:** This command does not apply to IBM TN3270 emulations.

## Arguments

### Text

The **Text** argument is the string the SET NETID command places into the NetID system variable. The strings “Tymnet” or “Telenet” are examples of a network ID.

## Comments

Compare with the NETID function.

## Example

This script segment:

```
Set NetID "NetWorld"  
PERFORM "Call"  
.  
.  
.  
PERFORM "Hangup"  
SET NETID "NetUS"  
PERFORM "Call"  
.  
.  
.
```

changes the NetID system variable before it connects to remote system. Changing the NetID system variable allows the `Call` subroutine to access sections of a network with the same script commands.

---

## SET OUTGOINGCR

---

### SET OUTGOINGCR Option

The SET OUTGOINGCR command specifies whether line feed characters should be added to outgoing carriage returns.



**Note:** This command does not apply to IBM TN3270 emulations.

### Arguments

#### Option

The Option argument is specified by either of the following keywords:

Keyword	Action
CRLF	Directs DCS to add line feeds to all outgoing carriage returns.
CR	Directs DCS not to add line feeds to outgoing carriage returns.



---

# SET PARITY

---

## SET PARITY Type

The SET PARITY command sets the desired parity type.



**Note:** This command does not apply to IBM TN3270 emulations.

## Arguments

### Type

The **Type** argument designates the type of parity to be used. It is specified by one of the following keywords:

Keyword	Action
EVEN	Directs DCS to set the parity bit to make the overall parity of the character packet even.
ODD	Directs DCS to set the parity bit to make the overall parity of the character packet odd.
MARK	Directs DCS to set the parity bit to one.
SPACE	Directs DCS to set the parity bit to zero.
NONE	Directs DCS to expect all of the databits to be used for data transmission, with no bit used for parity.

## Comments

If you choose a binary transfer protocol requiring eight data bits, and the parity is not set to NONE, DCS automatically adjusts this setting during the transfer without permanently modifying it.

## Example

This command:

```
SET PARITY EVEN
```

sets the parity option to even.


---

# SET PASSTHROUGH

---

## SET PASSTHROUGH State


The SET PASSTHROUGH command determines whether DCS sends data directly to the printer driver.

 **Note:** This command does not apply to IBM TN3270 emulations.

### Arguments

#### State

The State argument must be one of the following strings: AUTO, 1 (true), 0 (false). A string expression may also be used.

 **Note:** The State argument, which is a string, must be enclosed in quotes.

String	Action
TRUE	If TRUE, DCS sends the print jobs to the printer driver in passthrough mode, if the printer driver supports this mode. The strings YES or ON may also be used.
FALSE	If FALSE, DCS will send print jobs to the current Windows printer driver, which formats the file for a type of printer and then sends the file to the printer selected in the Printer Setup dialog or in the PRINT OPEN command. The strings NO or OFF may also be used.
AUTO	If AUTO, the passthrough mode will be determined as needed.

### Comments

Not all printer drivers are capable of passthrough printing.

### Example

In this example:

```
SET PASSTHROUGH "TRUE"  
PRINT FILE "PSCRIPT.TXT"
```

PSCRIPT.TXT contains PostScript code which is sent directly to the printer, rather than to the Windows printer driver.

---

## SET PASSWORD

---

### SET PASSWORD Text

The SET PASSWORD command changes the characters of the Password system variable.



**Note:** This command does not apply to IBM TN3270 emulations.

### Arguments

#### Text

The **Text** argument is the string which the SET PASSWORD command places into the Password system variable.

### Comments

This command and its system variable are similar in nature to the SET NETID and the NetID variable. Compare with the PASSWORD function.

---

## SET PHONENUMBER

---

### SET PHONENUMBER PhoneNumber

The SET PHONENUMBER command sets the phone number for the CONNECT command when the session uses a phone line.



**Note:** This command does not apply to IBM TN3270 emulations.

### Arguments

#### PhoneNumber

The PhoneNumber argument is a string specifying the phone number. DCS ignores any parentheses, dashes, or spaces in the string. However, DCS pauses for one second for each comma in the string.

### Comments

The phone number specified must be in a format acceptable to your modem. The SET PHONENUMBER command only has an effect when you have configured a session which uses the Direct Serial connector; it cannot be used with the Modem connector.

### Examples

In this example:

```
LOAD "COMPUSER"  
SET PHONENUMBER @S1  
DIAL
```

the session file COMPUSER is loaded and then a phone number is dialed. The phone number is determined by the contents of the settings variable @S1.

In this example:

```
SET PHONENUMBER `0-312-555-1222,,,,,,5554445`
```

DCS allows dialing through a long distance system that requires an access code. When dialed, the modem dials the phone number first, waits six seconds while the access code request is made, and then sends the access code 5554445.

---

# SET RESULT

---

## SET RESULT String

The SET RESULT command assigns the specified string value to the Result system variable.

### Arguments

#### String

The String argument specifies the value assigned to the Result system variable.

### Comments

The contents of the Result system variable may be modified by several methods. The SET RESULT command assigns a value to the Result system variable. Throughout the script reference, you will also see notations of commands, such as TABLE COPY, that assign a value to the Result system variable. If a task error occurs, DCS will assign a value to the Result system variable equivalent to the error message.

For a list of error numbers and messages, see **Appendix A Task Errors**.

### Example

In this example:

```
WHEN ERROR 3
BEGIN
SET RESULT "Run Time Error: " | RESULT ()
TASKERROR 2
END
```

a custom routine is established for handling run time errors is established. When a run time (level 3) error occurs, DCS appends the normal contents of the Result system variable (a level 3 error message) to the string "Run Time Error:" This new string is now stored as the Result system variable. The level 2 error handling routine is then called using the TASKERROR command. It will be passed the new Result system variable. These commands allow the script to handle a run time error as if it were a warning error.

---

## SET RETRY

---

### SET RETRY Boolean

The SET RETRY command tells DCS whether to dial a phone number again after an unsuccessful connection. (This only affects the modem; no notice will appear in a dialog box.)



**Note:** This command does not apply to IBM TN3270 emulations.

### Arguments

#### Boolean

The Boolean argument is specified by a Boolean expression or keyword. The Boolean keywords ON, YES, and TRUE evaluate to TRUE. The Boolean keywords OFF, NO, and FALSE evaluate to FALSE.

If the Boolean evaluates to TRUE, DCS redials a phone number continuously until a successful connection is made. If the Boolean evaluates to FALSE, DCS does not redial an unsuccessful connection.

### Example

These commands:

```
SET RETRY TRUE
DIAL '5551234'
```

dial 5551234 repeatedly until a connection is made.

---

## SET RETRYDELAY

---

### SET RETRYDELAY Delay

The SET RETRYDELAY command indicates the amount of time DCS waits before redialing. This command applies only to modems.



**Note:** This command does not apply to IBM TN3270 emulations.

### Arguments

#### Delay

The Delay argument is a numeric specifying the number of seconds to wait between successive redials.

### Comments

If a delay less than 30 seconds is specified, DCS delays for the default time of 30 seconds.

### Example

After dialing, these commands:

```
SET RETRY TRUE
SET RETRYDELAY 60
```

direct DCS to wait for 60 seconds before redialing. DCS only redials if no connection is made, and then continues to redial every 60 seconds until a connection is established.

---

## SET SENDDELAY

---

### SET SENDDELAY Delay

The SET SENDDELAY command specifies the amount of time for DCS to wait between transmitting each character when sending data to a host with the SEND command.

#### Arguments

##### Delay

The Delay argument is a numeric specifying the delay (measured in units of one-sixtieth of a second). If a numeric of less than zero units is specified, the Delay argument defaults to zero. If a numeric of greater than 30 units is specified, the Delay argument defaults to 30 (one-half of a second).

#### Example

This command:

```
SET SENDDELAY 15
```

directs DCS to wait for 0.25 seconds between each character it transmits with the SEND command.



---

# SET SIGNAL

---

## SET SIGNAL Boolean

The SET SIGNAL command is a setting for the DIAL command and specifies whether DCS should sound the system bell when a connection is made.



**Note:** This command does not apply to IBM TN3270 emulations.

### Arguments

#### Boolean

The Boolean argument is specified by a Boolean expression or keyword. The Boolean keywords ON, YES, and TRUE evaluate to TRUE. The Boolean keywords OFF, NO, and FALSE evaluate to FALSE.

If the Boolean evaluates to TRUE, the bell sounds when a successful connection is made. If the Boolean evaluates to FALSE, no signal is made to indicate a successful connection.

### Example

In this example:

```
LOAD "SET1"  
SET SIGNAL TRUE  
DIAL "5551234"
```

the SET SIGNAL command directs DCS to ring the bell when a connection is made.

---

# SET SOUND

---

## SET SOUND Boolean

The SET SOUND command controls whether warning bells from the host are enabled.

### Arguments

#### Boolean

The Boolean argument is specified by a Boolean expression or keyword. The Boolean keywords ON, YES, and TRUE evaluate to TRUE. The Boolean keywords OFF, NO, and FALSE evaluate to FALSE.

If the Boolean evaluates to TRUE, warning bells are enabled. If the Boolean evaluates to FALSE, warning bells are disabled.

### Comments

This command disables warning bells that come into the session window only. It does not disable DCS's system warnings.



Also see: GENERALCONFIG command

### Example

This command:

```
SET SOUND FALSE
```

directs DCS not to sound incoming bells from the host system.


---

## SET STOPBITS

---

### SET STOPBITS Numeric

The SET STOPBITS command sets the interval of time between each character packet sent during a serial transmission. This interval signals the end of a character packet.

 **Note:** This command does not apply to IBM TN3270 emulations.

### Arguments

#### Numeric

The Numeric argument can be one of the following values:

Value	Action
1	Directs DCS to expect 1 stop bit per character packet.
1.5	Directs DCS to expect 1.5 stop bits per character packet.
2	Directs DCS to expect 2 stop bits per character packet.

### Example

In this example:

```
SET DATABITS 7
SET STOPBITS 1
SET PARITY MARK
```

the contents of character packets is established as seven data bits and one stop bit.

---

# SET TERMCLOSE

---

## SET TERMCLOSE Boolean WINDOW WinHandle

The SET TERMCLOSE command specifies whether you will be allowed to close the default session window during script execution.

### Arguments

#### Boolean

The Boolean argument is specified by a Boolean expression or keyword. The Boolean keywords ON, YES, and TRUE evaluate to TRUE. The Boolean keywords OFF, NO, and FALSE evaluate to FALSE.

If the Boolean argument evaluates to TRUE, you are allowed to close the session window during script execution. If the Boolean argument evaluates to FALSE, you are not allowed to close the session window during script execution until a SET TERMCLOSE TRUE command is executed.

#### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

The **WINDOW** clause directs DCS to control whether you are allowed to close the session window specified by the **WinHandle** argument.

### Comments

When the session window cannot be closed, DCS cannot be closed.

If a window handle is not specified, the command is applied to the active session window.

### Example

In this example:

```
SET TERMCLOSE FALSE
PERFORM gather_info
SET TERMCLOSE TRUE
```

you are not allowed to close a session window while DCS is executing the `gather_info` procedure.

---

# SET TERMFONT

---

## SET TERMFONT FontName SizeX SizeY WINDOW WinHandle

The SET TERMFONT command changes the size of the font displayed in the session window.

### Arguments

#### FontName

The FontName argument no longer has an effect in the session window. This argument remains in the SET TERMFONT command for backward compatibility. DCS automatically chooses one of its fonts based on the terminal emulation you have chosen for the session.

#### SizeX SizeY

The SizeX and the SizeY arguments are integers specifying a valid pixel width and height for the font.

#### WINDOW WinHandle

The optional WINDOW clause includes the WINDOW keyword and the WinHandle argument. The WinHandle argument is an integer identifying a particular session window.

The WINDOW clause directs DCS to change the size of the font in the session window specified by the WinHandle .

### Comments

If a window handle is not specified, the command is applied to the active session window.

Wherever possible use the DISPLAYCONFIG command instead of this command.

### Example

This command:

```
SET TERMFONT ' ' 8 12
```

sets the session font to eight pixels wide and twelve pixels high.

---

## SET USERID

---

### SET USERID Text

The SET USERID command changes the characters of the UserID system variable.



**Note:** This command does not apply to IBM TN3270 emulations.

### Arguments

#### Text

The **Text** argument is the string that the SET USERID command places into the UserID system variable.

### Comments

This command and its system variable are similar in nature to the SET NETID command and the NetID system variable. Compare with the USERID function.


---

## SET WILDCARD

---

### SET WILDCARD Str1 Str2

The SET WILDCARD command changes the characters that normally represent arbitrary characters in a string.

 **Note:** This command does not apply to IBM TN3270 emulations.

#### Arguments

##### Str1

The first character of the **Str1** argument becomes the character that represents any single arbitrary character. The question mark (?) normally represents this character.

##### Str2

The first character of the optional **Str2** argument becomes the character that represents any number of arbitrary characters, including no (zero) characters, or a null character. The asterisk (\*) normally represents this character.

#### Comments

When using a command or function where the question mark or asterisk must be literal characters rather than wildcard characters, use the SET WILDCARD command to assign a different character to the wildcard as shown in the first example below.

#### Example

In this example:

```
CONNECT
SET WILDCARD "?" "t"
WAIT STRING "****"
SEND $password
SET WILDCARD "?" "*"
```

DCS sets the wildcard that represents any arbitrary string to t, so the WAIT STRING command can search for the asterisk (\*) character. After sending the \$password string, DCS resets the wildcard character to its normal value.

---

## SET WINDOWTITLE

---

### SET WINDOWTITLE **String** WINDOW WinHandle

The SET WINDOWTITLE command changes the string displayed in the active window's title bar to the specified string.

#### Arguments

##### String

The String argument specifies the string to be displayed.

##### WINDOW WinHandle

The optional WINDOW clause includes the WINDOW keyword and the WinHandle argument. The WinHandle argument is an integer identifying a particular child window.

The WINDOW clause changes the title of the window to the string specified by WinHandle .

#### Comments

If a window handle is not specified, the command is applied to the active window.

#### Example

In this example:

```
CONNECT "SCR1"  
SET WINDOWTITLE "Host Session1"
```

DCS opens, and connects, to a session window with the name SCR1. DCS then changes the title of the session window to Host Session1.



---

# SET WORDWRAP

---

## SET WORDWRAP Column

The SET WORDWRAP command is used in conjunction with the LOGTOFILE command to specify the column at which to wrap words.

 **Note:** This command does not apply to IBM TN3270 emulations.

### Arguments

#### Column

The Column argument is a numeric (from 0 to n) that specifies the desired column. If the value of Column is 0 (zero), wordwrap is deactivated.

### Example

In this example:

```
SET WORDWRAP 95
LOGTOFILE "ONLINE.TXT"
PERFORM get_data
FILE CLOSE "ONLINE.TXT"
```

the word wrap is set to column 95. The LOGTOFILE command directs DCS to open a file into which the online session will be captured. After the data is received, the file is closed and saved.

---

## SET XCLOCK

---

### SET XCLOCK Time WINDOW WinHandle

The SET XCLOCK command specifies how long DCS will wait, while sending data to an IBM 3270 system, for an XCLOCK message from the host to be cleared before generating an execution error.

#### Arguments

##### Time

The Time argument is a string in the following format:

Hours:Minutes:Seconds

Hours and minutes are optional, but if hours are specified, minutes must also be specified.

##### WINDOW WinHandle

The optional WINDOW clause includes the WINDOW keyword and the WinHandle argument. The WinHandle argument is an integer identifying a particular session window.

#### Comments

To determine if a run time error has occurred, use the ERROR( ) function or WHEN ERROR command.

If a window handle is not specified, the command is applied to the active session window.

#### Example

In this example:

```
SET XCLOCK "1:00"  
SEND ""
```

DCS waits for one minute for the XCLOCK state to be cleared on the host after the SEND command is executed.

---

# SET XSYSTEM

---

## SET XSYSTEM Time WINDOW WinHandle

The SET XSYSTEM command specifies how long DCS should wait, when sending data to an IBM 3270 system, for an XSYSTEM message from the host to be cleared before generating an execution error.

### Arguments

#### Time

The Time argument is a string in the following format:

Hours:Minutes:Seconds

Hours and minutes are optional, but if hours are specified, minutes must also be specified.

#### WINDOW WinHandle

The optional WINDOW clause includes the WINDOW keyword and the WinHandle argument.

The WinHandle argument is an integer identifying a particular session window.

### Example

In this example:

```
SET XSYSTEM "2:00"  
SEND " "
```

an execution error will occur if an XSYSTEM message from the host is not cleared within two minutes of sending the space character.

---

# SETTINGS

---

## SETTINGS Tab WINDOW WinHandle

The **SETTINGS** command opens the **Session Properties** dialog, allowing you to modify an option on a tab within the dialog before resuming script execution.

### Arguments

#### Tab

The **Tab** argument is specified by one of the following keywords:

Keyword	Action
COMMUNICATIONS	Opens the Connectors tab of the Session Properties dialog.
EMULATE	Opens the Emulations tab of the Session Properties dialog.
MOUSE	Opens the Mouse tab of the Session Properties dialog.
TRANSFERS	Opens the File Transfers tab of the Session Properties dialog.

#### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

### Comments

The **ERROR** function returns **TRUE** if you select the **Cancel** button in any of the dialogs this command presents.

If a window handle is not specified, the command is applied to the active session window.

If no session window is open, DCS will open a new session and default to the Connectors tab regardless of the **Tab** keyword.

### Example

In this example:

```
SETTINGS COMMUNICATIONS
```

the **Connectors** tab of the **Session Properties** dialog is displayed on the screen, allowing you to modify the communication settings during script execution.

---

# SHOW

---

## SHOW

The SHOW command directs DCS to display script commands in a dialog box as they execute.

### Arguments

The SHOW command takes no arguments.

### Comments

The DCS compiler resolves the SHOW command at compile time. The SHOW command displays the commands that execute from the routine in which you have placed the SHOW command. Therefore, you must insert the SHOW command into each subroutine you want to debug.

The NOSHOW command will cancel the SHOW command.

### Example

See the DEBUG command.

---

# SPAWN

---

## SPAWN SAFE Target WINDOW WinHandle

The SPAWN command instructs DCS to start another script. The spawned script runs independently of the script which spawned it. The original script also continues to execute without interruption.

### Arguments

#### SAFE

The optional **SAFE** keyword prevents the target script from starting if it would be attached to a default session window handle that already has at least one script attached to it. In this way the SPAWN command could avoid starting a competing script in a session window to which a script is already attached.

#### Target

The **Target** argument is a string specifying the script to start.

#### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular window in DCS.

The **WINDOW** clause directs DCS to assign the specified window handle as the new script's default session handle.

### Comments

The SPAWN and PERFORM commands are similar. The SPAWN command provides different functionality, however. With PERFORM, execution of the calling script is interrupted until the **Target** of the PERFORM command completes execution and returns. With the SPAWN command, the **Target** script executes completely independently and asynchronously.

### Example

In this example:

```
$DCscript = "dodde.dct"  
%hndle = WINDOWHND("Session1")  
SPAWN SAFE $DCscript WINDOW %hndle
```

This example causes the script `dodde.dct` to run and act upon `Session1` only if no other script is already attached to that session.

---

# SWITCH

---

**SWITCH SwitchVariable**

**CASE Var1:**

**Command**

**LEAVE**

**CASE Var2:**

**Command**

**LEAVE**

**DEFAULT:**

**Command**

**LEAVE**

**SWITCH END**

The SWITCH command allows script execution to branch to multiple command blocks.

## Arguments

### SwitchVariable

The SwitchVariable argument is a numeric or string variable or expression that is compared to each CASE variable.

### CASE Varn

**Command**

**LEAVE**

Each CASE clause compares the Varn argument with the SwitchVariable. The **Command** argument is a command or command block which DCS executes if the variables match. If the variables do not match, DCS compares the SwitchVariable to the next CASE variable. If an optional LEAVE command is inserted, execution branches past SWITCH END to the line following execution of the **Command** argument.

**DEFAULT:**

**Command**

**LEAVE**

DCS executes the optional **DEFAULT** clause and Command block if none of the CASE clauses match the SwitchVariable.

---

## SWITCH, *continued*

---

### Example

In this example:

```
SWITCH $char
CASE "A":
DISPLAY "Apple^M"
LEAVE
CASE "B":
DISPLAY "Ball^M"
LEAVE
CASE "C":
CASE "D":
DISPLAY "Candy/Dog^M"
LEAVE
DEFAULT:
DISPLAY "none^M"
LEAVE
SWITCH END
```

DCS compares the contents of the string variable `$char` to each of the cases. If the contents match the string argument for one of the cases, the script displays the appropriate string. For example, if `$char` contains "B", the script displays "Ball" followed by a carriage return.



---

# SYSTEM

---

## SYSTEM SysNum P1, P2, ...

The SYSTEM command provides script access to operating system-level parameters and environment variables.



**Note:** This command does not apply to IBM TN3270 emulations.

### Arguments

#### SysNum

The SysNum argument is specified by a hexadecimal value from the table below.

#### P1, P2, ...

The optional *P<sub>n</sub>* arguments are specified by one or more integers or strings.

SysNum	Definition
0x00FF	<p>The command sends a string to a debugging terminal. The first parameter for the command is a string that you want to send to the debugging terminal. The second parameter is a null, or empty, string. To use this command you must run DBWIN.EXE. For further information about debugging terminals and DBWIN.EXE, see the Microsoft Windows Software Development Kit (SDK) and the discussion of the OutputDebugString function in the SDK.</p> <pre>SYSTEM 0x00FF "Procedure succeeded." ""</pre>
0x0701	<p>This command executes the Windows WinHelp function and opens a help file at a topic defined by a context number of the help file. The context number must be contained in the Map section of the help file's HPJ or in one of the HPJ's support files.</p> <p>The first parameter for this command is a string parameter, which is the path name of the help file.</p> <p>The second parameter for this command is an integer, indicating the Windows help function that you would like to access. These numbers are located in WINDOWS.H, which is a part of the Microsoft Windows Software Development Kit (SDK).</p> <p>The third parameter is an integer, which is the context number of the topic that you want to display.</p> <p>Do not use pointers or custom data types with this command.</p> <pre>SYSTEM 0x0701 \$FileName \$WinFuncNum %ContextNum</pre>
0x0702	<p>This command executes the Windows WinHelp function and opens a help file at a topic where WinHelp finds the first instance of a keyword (or phrase) in the help file.</p> <p>The first parameter for this command is a string parameter, which is the path name of the help file.</p> <p>The second parameter for this command is an integer, indicating the Windows help function that you would like to access. These numbers are located in WINDOWS.H, which is a part of the SDK.</p> <p>The third parameter is a string, which is a keyword in the help file.</p> <p>Do not use pointers or custom data types with this command.</p> <pre>SYSTEM 0x0702 \$FileName %WinFuncNum %ContextNum</pre>

---

## SYSTEM, *continued*

---

SysNum	Definition
0x0803	<p>This command changes the font used in the previous dialog in the script.</p> <p>The first parameter is a string specifying the title of the dialog box or an empty string.</p> <p>The second parameter identifies a particular dialog control as specified by one of the following string values: "ICON", "MESSAGE", "BUTTON", "ICONBUTTON", "RADIOGROUP", "RADIOBUTTON", "CHECKBOX", "EDITTEXT", "LISTBOX", "GROUPBOX"</p> <p>The third parameter identifies a particular control as specified by a control index number. This follows the same syntax as the existing script dialog verbs ("Edittext" 1, "Message" 5, etc.).</p> <p>The fourth parameter is either the string "Fixed," or "Variable." Other values are ignored. "Fixed" will assign the system Fixed-pitch font to the specified control. "Variable" will assign the System Variable-pitch font to the control. Note that in Windows 3.0 and above, the default font for dialog controls is the System Variable font.</p> <p>The fifth parameter is optional and identifies a particular dialog as specified by a dialog index number.</p> <pre>SYSTEM 0x0803 \$DlgTitle \$control %index \ \$font [%dlgIndex]</pre>
0x0900	<p>This command sets the font size for the next dialog in the script to 8 point.</p> <pre>SYSTEM 0x0900</pre>
0x0901	<p>This command changes the title in a dialog box created by a DCS script.</p> <p>For this command to affect the title bar of the dialog box, the dialog box must have been defined with a title bar, and the title must be something other than an empty or null string.</p> <pre>SYSTEM 0x0901 "Changed Title Name"</pre>

---

### Comments

The parameters listed in this electronic document are always the most up-to-date. However, please note that the parameters for this command are not universal between versions of DCS and are subject to change without notice.

Contact FutureSoft Technical Support for information or questions about the SYSTEM command.



Also see: SYSTEM function

### Example

See each entry in the table above for a syntax example.

---

## TABLE CLEAR

---

### TABLE CLEAR Table

The TABLE CLEAR command clears any existing data from a specified table.

#### Arguments

##### Table

The Table argument is a numeric specifying the table number.

#### Comments

Because structured tables are maintained in memory, clearing a table makes more memory available.

#### Example

In this example:

```
TABLE DEFINE 0 FIELDS CHAR 20 CHAR 10
TABLE LOAD 0 FROM "EXPENSES" AS SYLK
PERFORM modifyData
TABLE SAVE 0 TO "EXPENSES" AS SYLK
TABLE CLEAR 0
```

a structured table is defined and data loaded into it. When the table has been modified, it is saved to a file and cleared from memory.

---

## TABLE CLOSE

---

### TABLE CLOSE Table

The TABLE CLOSE command closes a table.

#### Arguments

##### Table

The Table argument is a numeric specifying the table number.

#### Example

In this example:

```
TABLE DEFINE 5 TEXT "MYFILE.TXT"  
@R5 = "New Line"  
RECORD WRITE 5  
TABLE CLOSE 5
```

the TABLE CLOSE 5 statement closes Table 5 (five) after the script writes the string "New Line" into MYFILE.TXT.

---

# TABLE COPY

---

## TABLE COPY SourceTable TO DestTable Contents String

The TABLE COPY command copies the contents of the specified source table to the specified destination table.

### Arguments

#### SourceTable

The SourceTable argument is a numeric (from 0 to 15) specifying the table number of the source table.

#### DestTable

The DestTable argument is a numeric (from 0 to 15) specifying the table number of the destination table.

#### Contents String

The optional **Contents** argument is one of the following keywords:

Keyword	Action
INCLUDE	Directs DCS to copy only those records that begin with the specified String.
EXCLUDE	Directs DCS to copy all records except those that begin with the specified String.
RANGE	Specifies a range of characters or fields to be copied from the specified records. You must follow the RANGE keyword with a beginning record or byte number and an ending record or byte number.

---

#### ▼ Structured Tables

The RANGE values are record numbers. Remember that DCS begins numbering the first record in a structured table and the first byte in a text table with zero.

#### ▼ Text Tables

The RANGE values are characters positions or byte numbers in the table. A record may have a maximum of 254 characters. When performing a RANGE copy from a text table, the last full line is always copied to the destination table.

### Comments

The Result system variable is set to the number of records copied.

If the destination table does not exist, the new table is created. If an existing table is replaced, any data in the existing table is overwritten.

Both source and destination tables must have the same format.

---

## TABLE COPY, *continued*

---

### Example

In this example:

```
TABLE COPY 1 TO 2 EXCLUDE "_"  
DISPLAY RESULT () | " records copied"
```

the records from Table 1 (one) are copied to Table 2 (two), except for underscore characters, and then displays the number of records copied.

In this example:

```
TABLE DEFINE 0 TEXT "test.TXT"  
TABLE DEFINE 1 TEXT "testout.TXT"  
TABLE COPY 0 TO 1 RANGE 3 9  
TABLE CLOSE 0
```

the characters in positions 3 (three) through 9 (nine) are copied from Table 0 (zero) to Table 1 (one).

---

## TABLE DEFINE

---

### **TABLE DEFINE Table FIELDS f1...fi FILE TEXT FileName**

The TABLE DEFINE command creates one of two types of tables.

Tables allow for manipulation of whole files or masses of data. Two types of tables are possible: structured or text. Structured tables have fixed length records of fixed length fields. Text tables have records of any number of bytes from 1 to 254.

### **Arguments**

#### **Table**

The table argument is an integer (from 0 to 15) specifying the table number. All references to a table will reference this number.

#### **▼ Structured Tables**

##### **FIELDS f1...fi**

The FIELDS clause defines the record layout for the table data (and corresponding record buffer). Each fi argument includes:

- 1 One of the following keywords specifying the logical data type for the field:

CHAR          INT          REAL

- 2 An integer specifying the length of the field

A field may have a maximum length of 254 bytes. A table may have a maximum of 254 fields in a record (numbered 1 to 254). A table definition determines the number of fields in a table's record.

##### **FILE**

The optional **FILE** keyword temporarily stores the data as a file-based table on disk (for tables 64K in size, or larger) until the script ends or until DCS executes a **TABLE CLOSE** command. A table can be saved to a permanent file with the **TABLE SAVE** command. If you have not included the **FILE** keyword, the table is maintained in memory.

#### **▼ Text Tables**

##### **TEXT FileName**

The **FileName** argument specifies the file which fills the table. Any changes made to the table are reflected in changes to the file. If the file does not exist, an empty table is created on the computer hard drive.

### **Comments**

All data maintained in a file is stored as a string. The CHAR, INT, and REAL data type specifications maintain the logical data type for data import and export to other applications.

Fields specified as either INT or REAL are maintained as right-justified fields.

DCS cannot directly access table data. It must access one record at a time through the corre-

---

## TABLE DEFINE, *continued*

---

sponding record buffer variable `@Rn`, where `n` is the table number. Table data is read with the `RECORD READ` command and written to a table with the `RECORD WRITE` command.

DCS clears the contents of a currently defined table when the script executes another `TABLE DEFINE` command with the same number as the currently defined table.

A record in a table may contain a maximum of 254 characters.

### Example

In this example:

```
TABLE DEFINE 1 FIELDS CHAR 10 INT 5 SAVE DATA
```

Table 1 (one) is defined as a file-based structured-record table, with two fields per record. Field 1 (one) is a character type and has ten characters. Field 2 (two) is an integer type and has five characters.

In this example:

```
TABLE DEFINE 1 TEXT "MEMO.TXT"

*ReadLoop
RECORD READ 1 at 0
WHILE NOT EOF ()
BEGIN
DISPLAY @R1 | "^M"
RECORD READ 1
END
TABLE CLOSE 1
```

Table 1 (one) is defined as a text table containing characters from `MEMO.TXT`. The script then presents the text of the table, one line at a time, in the active session window.

In this example:

```
TABLE DEFINE 1 TEXT "BUDGET.XLS"
```

the `TABLE DEFINE` command redefines Table 1 (one) as a text table.



---

# TABLE LOAD

---

## TABLE LOAD Table FROM Source AS Format

The TABLE LOAD command imports data from a file or the clipboard to a structured table.



**Note:** This command is used only with structured tables.

### Arguments

#### Table

The **Table** argument is an integer (from 0 to 15) specifying the number of the table into which to load the data.

#### FROM Source

The **FROM** clause specifies the source of the data. The value of the **Source** argument is either the keyword **CLIPBOARD**, or the name of a file. If the **Source** argument specifies a data file, a valid file name must be used.

#### AS Format

The **AS** clause specifies the standard data format in which to store the imported data. The **Format** argument is one of the following keywords:

DIF      DYNACOMM      SYLK      TEXT

The **DYNACOMM** format is valid only for file-based tables and cannot be used for data interchange (see the **TABLE DEFINE** command).

### Example

This command:

```
TABLE LOAD 5 FROM "FORECAST.SYL" AS SYLK
```

loads the contents of **FORECAST.SYL** into table five in **SYLK** format.


---

## TABLE SAVE

---

### TABLE SAVE Table TO Destination AS Format

The TABLE SAVE command exports the contents of a structured table to either the clipboard or a file.

 **Note:** This command is used only with structured tables.

#### Arguments

##### Table

The **Table** argument is an integer (from 0 to 15) specifying the table whose contents you want to save.

##### TO Destination

The **TO** clause specifies the destination of the table's data. The **Destination** argument is either the keyword CLIPBOARD, or the name of a file. If you use the **Destination** argument to specify a data file, it must be a string specifying a valid file name for your system.

##### AS Format

The **AS** clause specifies the standard data format in which to store the table's data. The **Format** argument is one of the following keywords:

DIF      DYNACOMM      SYLK      TEXT

The DYNACOMM format is valid only for file-based tables and cannot be used for data interchange.

#### Example

In this example:

```
WHEN POKE 0 TABLE 0 "item0"  
BEGIN  
  DISPLAY "POKE DATA RECEIVED"  
  TABLE SAVE 0 TO "DATA" AS SYLK  
END
```

when a **POKE** is received from the DDE client on the topic item0, DCS displays the message "POKE DATA RECEIVED" in the session window and saves the received data to the file named DATA.

---

## TABLE SORT

---

### TABLE SORT Table Fld1 Dir1 Fld2 Dir2 Fld3 Dir3

The TABLE SORT command sorts the contents of a structured table based upon a sort criteria.



**Note:** This command is used only with structured tables.

#### Arguments

##### Table

The **Table** argument is a numeric (from 0 to 15) specifying the table to sort.

##### Fld*n* Dir*n*

Each **Fld*n* Dir*n*** argument pair defines a sort criterion. The **Fld*n*** argument is an integer specifying the desired field number. The **Dir*n*** argument is specified by one of the following keywords:

ASCEND      DESCEND

The **ASCEND** keyword performs the sort in ascending order. The **DESCEND** keyword performs the sort in descending order.

#### Comments

A maximum of three sort keys can be defined. The second sort key is used only when there are two or more identical items from the first sort key.

#### Example

This command:

```
TABLE SORT 0 1 ASCEND 5 DESCEND
```

sorts Table 0 (zero) using two criteria. It is first sorted in ascending order using the contents of field 1 (one). If there are two or more identical items from the first sort, a second sort is performed in descending order using the contents of field 5 (five).

---

# TASKERROR

---

## TASKERROR Level Code

The TASKERROR command calls the DCS internal execution error-handling routine and passes the error level and code.

### Arguments

#### Level

The optional **Level** argument is a numeric, from 0 (zero) to 4 (four) that specifies the desired error level. The following error levels are available:

Level	Meaning
0 (zero)	Fatal
1 (one)	Critical
2 (two)	Warning
3 (three)	Run time
4 (four)	User-defined



**Note:** If a Level argument is not included, error level 3 (three) is used.

#### Code

The optional **Code** argument is a numeric specifying the three-digit error code that identifies the type of error. The error message displayed depends on the code. If you do not include the **Code** argument, three zeros (000) is passed as the code. If you include the **Code** argument you must include the **Level** argument.

### Comments

This command can simulate the occurrence of an execution error of a type you specify. See **Appendix A Task Errors**.

All error codes outside of the range 100-999 produce the string “User-defined error”.

Error level 3 (three - run-time) does not cause the display of an error message. To display an error message, the script must trap a level 3 error with a **WHEN ERROR 3** command (see the example).



Also see: **SET RESULT** command

---

## TASKERROR, *continued*

---

### Example

```
;set traps to handle errors
when error 0 perform HandleError(0)
when error 1 perform HandleError(1)
when error 2 perform HandleError(2)
when error 3 perform HandleError(3)

;simulate three user-defined errors
taskerror 0 221 ; a "real" error
taskerror 1 221 ; a "real" error
taskerror 2 221 ; a "real" error

taskerror ; default
taskerror 3 ; default
taskerror 3 99 ; custom

cancel

;error trap routine
*HandleError(%level)
    $level=str(%level)
    $result=result()
    dialog "Error Handler - Level " | $level
        message "Error Text:"
        message $result
        button default "OK" resume
    dialog end
    wait resume
    dialog cancel
return
```

---

# TASKSTOP

---

## TASKSTOP taskID

The TASKSTOP command will stop the script with the given taskID.

### Arguments

#### taskID

The argument is a numeric, specifying the desired script to stop.

### Comments

Use the menu option Script: Status... to see a list of running scripts.

See also the SPAWN command.

See also the TASKNAME function.

See also the TASKFILE function.

See also the TASKLIST function.

### Example:

```
spawn "script2"
spawn "simpledlg"
spawn "dialogupdate"
dialog (7,7) "Stop all other scripts"
  message "tasklist _____"
  message "taskid _____"
dialog end
$meTask = TASKLIST(-1)
$allTasks = TASKLIST()
dialog update message 1 $allTasks
WHILE($allTasks != "")
BEGIN
  parse $allTasks $tID "," $allTasks
  dialog update message 2 $tID | " " | taskname(NUM($tID))
  wait delay "2"
  if($tID <> $meTask)
    TASKSTOP num($tID)
END
dialog update message 1 "Nobody but me!"
wait delay "9"
cancel
```

---

# TIMER RESET

---

## TIMER RESET Timer

The `TIMER RESET` command directs DCS to set a timer to zero.

### Arguments

#### Timer

The `Timer` argument is a numeric from 0 (zero) to 3 (three) that specifies a timer to reset. Four separate timers can be maintained in a single script. These timers can only be accessed through a script.

### Example

In this example:

```
TIMER RESET 0
CONNECT
PERFORM login1
DISPLAY "Connected to login 1 for " | TIMER (0)
TIMER RESET 0
```

the timer is reset before the login. By using the `TIMER ( )` function, the script provides the length of the login process or how long DCS has been connected to the remote system up to that point in the script.

---

# TITLE

---

## TITLE String

The TITLE command assigns a title to a script. While the script executes, the String argument displays in the title bar of the DCS application window.

### Arguments

#### String

The String argument specifies a title. A literal string of characters enclosed within quotation marks must be used, not a string variable. The maximum length of the String argument is 61 characters.

### Comments

Only one title may be assigned to each task file.



Also see: `Set AppTitle` command

`Set WindowTitle` command

### Example

When DCS executes this script segment:

```
TITLE "My Script"
```

the text in the DCS window changes to "My Script".



---

# TOOLBARHIDE

---

## TOOLBARHIDE BarName

The TOOLBARHIDE command closes the specified toolbar, removing it from view in the application window.

### Arguments

#### BarName

The BarName argument is used to specify which toolbar to hide. It must be one of the following toolbars:

ALL	FILE	SESSION	TRACE	OLE
STANDARD	EDIT	TRANSFER	SCRIPT	

or the name assigned to a user-defined toolbar.

### Comments

Toolbars hidden via scripting will remain hidden, even after DCS has been restarted, until shown again.



Also see: TOOLBARSHOW command

### Example

This line of script:

```
TOOLBARHIDE "transfer"
```

hides the Transfer toolbar. If the toolbar is already hidden, the command has no effect.

---

# TOOLBARSHOW

---

## TOOLBARSHOW BarName

The TOOLBARSHOW command opens the specified toolbar, displaying it in the toolbar area of the application window.

### Arguments

#### BarName

The BarName argument is used to specify which toolbar to show and must be one of the following:

ALL	FILE	SESSION	TRACE	OLE
STANDARD	EDIT	TRANSFER	SCRIPT	

or the name assigned to a user-defined toolbar.

### Comments

Toolbars displayed via scripting will remain visible, even after DCS has been restarted, until hidden.

Also see: TOOLBARHIDE command

### Example

This line of script:

```
TOOLBARSHOW "transfer"
```

displays the transfer toolbar. If the toolbar is already visible, the command has no effect.

---

## TRANSFERS

---

**TRANSFERS** CommandProcessor BLOCKSIZE Status Length ISSUECLEAR Status PACKETSIZENumber HOSTPROGRAM FileName RECORDLENGTH Status Length RECORDFORMAT Type SPACE Status InitialSpace AddedSpace UNITS Type CPUUSAGE Number TIMEOUT Number HOSTCODEPAGE Country PCCODEPAGE Country WINDOW WinHandle

The TRANSFERS command sets the parameters in which DCS will send and receive files using a remote computer and its command processor. This command only sets the parameters for file transfers involving the IND\$File file transfer program, or similar mainframe transfer programs.

### Arguments

#### CommandProcessor

The optional **CommandProcessor** argument is either the CMS or TSO keyword.

#### BLOCKSIZE Status Length

Include the optional **BLOCKSIZE** clause to transfer a file with a remote system with a TSO command processor.

The **BLOCKSIZE** clause includes the keyword **BLOCKSIZE** and the **Status** clause. The **Status** clause includes either:

Clause	Action
ON Length	Specifies to use the value of the Length argument to determine the block length of the host data set.  The Length argument is an integer and determines the size of the host data set in units of bytes.  The default value for a data set block size is 80.
OFF	Specifies that the default length associated with the BLOCKSIZE clause is to be used to determine the size for a unit of space allocated with the SPACE option of the TRANSFERS command (when you have not included the UNITS clause in the command).

#### ISSUECLEAR Status

Include the optional **ISSUECLEAR** clause to transfer a file with a remote system with a VM/CMS command processor.

The **ISSUECLEAR** clause includes the **ISSUECLEAR** keyword and the **Status** keyword of either:

Keyword	Action
ON	Directs the remote system to clear the terminal screen before a file transfer is initiated.
OFF	Directs that the remote system should not automatically clear the terminal screen before a transfer.

---

## TRANSFERS, *continued*

---

### PACKETSIZE Number

The optional **PACKETSIZE** clause specifies the size of data packets transferred to and from the remote system command processor.

The **PACKETSIZE** clause includes the keyword **PACKETSIZE** and the **Number** argument. The **Number** argument is an integer (from 0 to 32). However, if the file transfer is a non-SNA transfer, the maximum packet size is 7 (seven). SNA transfers can be as large as 32. Transfers with an old version of the IND\$FILE transfer program should use a packet size of 2 (two).

### HOSTPROGRAM FileName

The optional **HOSTPROGRAM** clause allows access to a variety of file transfer programs on a remote system.

The **HOSTPROGRAM** clause is composed of the **HOSTPROGRAM** keyword and the **FileName** argument. The **FileName** argument is a string of a valid file name for a file transfer program on the remote system.

The default file transfer program for DCS is IND\$FILE.

### RECORDLENGTH Status Length

Include the optional **RECORDLENGTH** clause when a file is transferred with a remote system that has a TSO or VM/CMS command processor. The **RECORDLENGTH** clause is composed of the **RECORDLENGTH** keyword and the **Status** clause.

The **Status** clause includes either:

Clause Keywords	Action
ON Length	Specifies that the remote system is to use the value of the Length argument to determine the length of a logical record in a host data set. The Length argument is an integer that determines the size of a record in units of bytes. If a file on the remote system is being appended or replaced, the remote system ignores the <b>RECORDLENGTH</b> clause or its default. On a remote system with a TSO command processor, the default length of a fixed or undefined length record is 80 bytes; however, the default for a variable length record is 84 bytes. On a remote system with a VM/CMS command processor, the default length of a fixed length record is 80 bytes, and the default length of a variable length record is 84 bytes.
OFF	Specifies that the remote system is to use the default record length associated with the particular command processor and record format involved in the transfer.

---

## TRANSFERS, *continued*

---

### RECORDFORMAT Type

Include the optional **RECORDFORMAT** clause to send a file to a remote system with a TSO or VM/CMS command processor.

The **RECORDFORMAT** clause includes the **RECORDFORMAT** keyword and the **Type** argument. The **Type** argument may be one of the following keywords:

Files Sent to	Possible Keywords
TSO command processor	FIXED, VARIABLE, or UNDEFINED
VM/CMS command processor	FIXED or VARIABLE

### SPACE Status InitialSpace AddedSpace

Include the optional **SPACE** clause when a file is sent to a remote system that has a TSO command processor.

The **SPACE** clause includes the keyword **SPACE** and the **Status** clause. The **Status** clause includes either:

Clause Options	Action
ON InitialSpace AddedSpace	The ON keyword specifies to use the InitialSpace and AddedSpace arguments to determine the total amount of space allocated to a new data set. The InitialSpace argument is an integer that specifies the initial number of units allocated to a new data set. The AddedSpace argument is an integer that specifies the number of units of space added to the initial space (when the initial space fills with the new data set, and when the data set requires more space on the system). The maximum number of units that might be available on a remote system is [AddedSpace + (15 x InitialSpace)]. The units for the SPACE clause are determined by the UNITS clause of the TRANSFERS command.
OFF	The OFF keyword specifies that the remote system is to use default values for space allocation.

### UNITS Type

Include the optional **UNITS** clause when DCS sends a file to a remote system that has a TSO command processor. The **UNITS** clause includes the keyword **UNITS** and the **Type** argument.

The **Type** argument is specified by one of the following keywords:

BLOCKS      TRACKS      CYLINDERS

The **UNITS** clause determines the unit of space used by the **SPACE** clause.

---

## TRANSFERS, *continued*

---

### **CPUUSAGE** Number

The optional **CPUUSAGE** clause determines how long DCS controls the CPU on your PC during a file transfer.

The **CPUUSAGE** clause includes the keyword **CPUUSAGE** and the **Number** argument. The **Number** argument is an integer between 1 (one) and 9 (nine), inclusive. The default value is 5 (five).

### **TIMEOUT** Number

The optional **TIMEOUT** clause sets the amount of time DCS waits for a remote system to respond to a transfer request.

The **TIMEOUT** clause includes the keyword **TIMEOUT** and the **Number** argument. The **Number** argument is an integer indicating an amount of time in units of seconds. The default **Number** argument value is 30 seconds.

### **HOSTCODEPAGE** Country **PCCODEPAGE** Country

The optional **HOSTCODEPAGE** and **PCCODEPAGE** clauses set the code pages for a file transfer via script. Code pages set with these options override previous session settings. The **Country** argument for either **HOSTCODEPAGE** or **PCCODEPAGE** is one of the following strings:

Australia	Belgium	Canada (English)
Canada (French)	Denmark	Finland
France	Germany	Italy
Latin America	Netherlands	Norway
Portugal	Spain	Sweden
Switzerland (French)	Switzerland (German)	United Kingdom
United States	International	ANSI

### **WINDOW** WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

The **WINDOW** clause directs DCS to set up transfer parameters for the session specified by the **WinHandle**.

---

## TRANSFERS, *continued*

---

### Comments

The parameters in this command determine the operation of a file transfer started by a file transfer command (FILE SEND BINARY or FILE RECEIVE BINARY). If you do not include the TRANSFERS command in a script before a file transfer command, the file transfer command uses the existing transfer parameters of the session or uses the relevant parameters defined prior to a previous file transfer command.

If the **WINDOW** clause is not included, DCS changes the transfer parameters for the active session window.

### Example

In this example:

```
TRANSFERS CMS \  
BLOCKSIZE OFF \  
ISSUECLEAR OFF \  
BUFFERSIZE 2 \  
HOSTPROGRAM "IND$FILE" \  
RECORDLENGTH OFF \  
RECORDFORMAT FIXED \  
SPACE OFF \  
UNITS BLOCKS \  
WINDOW 1234  
  
$PATH = DIRECTORY (SCRIPT) | "B.DCP"  
  
FILE SEND BINARY $PATH AS "TEST" "TASK" "A" \  
ASCII NOCRLF WINDOW 1234
```

the transfer environment is set with the TRANSFERS command, and then the file whose file name is in \$PATH, is sent to the remote system with the FILE SEND BINARY command.

---

## WAIT CHAR

---

### WAIT CHAR Character WINDOW WinHandle

The WAIT CHAR command pauses execution until the specified character is received in the session window.

 **Note:** The EDIT COPYSPECIAL command does not apply to the IBM TN3270 emulation.

#### Arguments

##### Character

The Character argument specifies a character for which to wait.

##### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

The **WINDOW** clause directs DCS to wait for character echo from a remote system in a specific session window.

#### Comments

If the window handle is not included, the command is applied to the active session window.

To wait until a host is in a ready state before sending data, see the special strings for use with the SEND command.

#### Example

This example:

```
DISPLAY "Please enter 'g' to go to next menu: ^M  
WAIT CHAR 'g'  
DISPLAY "Thank You. ^M"
```

works only in session windows using a terminal emulation other than 3270. The script example prompts you to press the [G] key. The WAIT CHAR command pauses the execution of the script until you press the [G] key (lower case letter g). After the [G] key is pressed, "Thank You." appears in the session window

This example:

```
DISPLAY "Execution Paused, Hit Control-D To Continue: ^M"  
WAIT CHAR CHR (0x04)
```

works only in session windows using a terminal emulation other than 3270. In this script fragment, the WAIT CHAR command pauses the execution of the script until a Control-D enters the session window. The hexadecimal number 0x04 represents Control-D in a script.



---

## WAIT CLOSE

---

### WAIT CLOSE WINDOW WinHandle

The WAIT CLOSE command pauses script execution until you close a text file opened by the LOGTO-FILE command.



**Note:** The EDIT COPYSPECIAL command does not apply to the IBM TN3270 emulation.

### Arguments

#### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular child window.

The **WINDOW** clause directs DCS to wait until the text file opened in a specific window is closed.

### Comments

If the **WinHandle** is not included, the command is applied to the active window.

---

## WAIT DELAY

---

### WAIT DELAY Time

The WAIT DELAY command pauses the execution of a script for a specified period of time.

#### Arguments

##### Time

The Time argument is a string in the following format:

hh:mm:ss.fff

where:

Time Element	Description
hh	Optional integer character string for a number of hours
mm	Optional integer character string for a number of minutes
ss	Required integer character string for a number of seconds
fff	Optional integer character string for a fraction of a second

For this command to pause a script for only a fraction of a second, the Time argument must have the following format:

0:0:0.fff

The format for pausing less than a second requires that you include a 0 (zero) for the hours, minutes, and seconds strings (with the colon between each) and that you include a period before the string for a fraction of a second.

#### Comments

A WAIT DELAY value of zero (WAIT DELAY “0”) is a special case of the WAIT DELAY command. This form of the command activates a WHEN command without entering a wait state. If a RESUME command executes after execution of WAIT DELAY “0”, execution of the script branches past the current WAIT command (the last WAIT command prior to execution of the WAIT DELAY “0” command, if any).

The script language allows this command to wait a fraction of a second.

---

## WAIT DELAY, *continued*

---

### Examples

This command:

```
WAIT DELAY "00:10"
```

directs DCS to pause script execution for ten seconds.

In this example:

```
WHEN TIMER "5" PERFORM CountSub
WAIT RESUME
DISPLAY "Aborted"
CANCEL

*CountSub
DIALOG
MESSAGE "Count:  "
BUTTON CANCEL "Cancel" RESUME
DIALOG END
SET %counter 1
WHILE (TRUE)
BEGIN
DIALOG UPDATE MESSAGE 1 "Count: " | STR (%counter)
INCREMENT %counter
WAIT DELAY "0"
END
```

a dialog box displays while a loop executes. Each time the `WAIT DELAY "0"` command executes, the script checks for the activation of a `WHEN` command. Because DCS treats active dialog controls as `WHEN` commands, DCS is able to determine if someone has selected the **Cancel** button during the execution of the loop.

Since the `WAIT DELAY "0"` command does not create a wait state, the execution of the `RESUME` command causes execution to cancel.

In this example:

```
WAIT DELAY "0:0:0.25"
```

the script waits for one quarter second.

---

# WAIT ECHO

---

## WAIT ECHO WINDOW WinHandle

The WAIT ECHO command pauses script execution until any character is received from the remote system.

### Arguments

#### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

The **WINDOW** clause directs DCS to wait for character echo from a remote system in a specific session window.

### Comments

If the **WINDOW** clause is not included in the command, DCS will wait for character echo in the active session window.

To wait until a host is in a ready state before sending data, see the special strings for use with the SEND command.

### Example

In this example:

```
WHEN QUIET `30`  
BEGIN  
  DISPLAY (0,0) "No Prompt"  
  CANCEL  
  END  
  WAIT ECHO  
  IF SEARCH (`ID:`) <> -1  
  SEND `USER`
```

DCS waits 30 seconds to receive input from the remote system. The SEARCH function then determines if the host sent an expected prompt.

---

## WAIT EDIT

---

### **WAIT EDIT (x, y, w, h) FileName**

The **WAIT EDIT** command invokes the memo editor and pauses script execution until you close the memo editor.

#### **Arguments**

##### **(x, y, w, h)**

The optional coordinate set **(x, y, w, h)** specifies the position and size of a memo window. It indicates the top left corner (x, y), the width (w), and the height (h).

##### **FileName**

The optional **FileName** argument is a string specifying the name of the file to edit. The **FileName** argument must specify a valid file name for your system.

If **FileName** is the string "?", DCS prompts for a file name during script execution.

If **FileName** is not provided or is the null string "", DCS opens a new memo window.

#### **Comments**

Closing the memo window resumes script execution at the line following the **WAIT EDIT** command. The document is saved automatically.

Executing a **RESUME** command is equivalent to closing the memo child window.

#### **Example**

In this example:

```
WAIT EDIT "bills.TXT"  
PERFORM file_transfer
```

the script opens the memo file **BILLS.TXT** and pauses script execution until the memo window is closed.

---

## WAIT PROMPT

---

### WAIT PROMPT NumChar WINDOW WinHandle

The WAIT PROMPT command pauses execution until a specified number of characters is received in a session window.



Note: The WAIT PROMPT command does not apply to the IBM TN3270 emulation.

#### Arguments

##### NumChar

The NumChar argument is an integer specifying a number of characters to receive.

##### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

The optional **WINDOW** clause directs DCS to wait for the characters in a particular session window.

#### Comments

If the **WINDOW** clause is not included in the command, DCS waits for the characters in the active session window.

DCS counts every character received.

To wait until a host is in a ready state before sending data, see the special strings for use with the SEND command.

#### Example

In this example:

```
WAIT PROMPT 10
PERFORM trial_run
```

script execution pauses until ten characters are received in the session window, then `trial_run` is performed.

---

## WAIT QUIET

---

### WAIT QUIET Time WINDOW WinHandle

The WAIT QUIET command pauses the execution of a script for a specified period of time wherein no character transmission occurs.

#### Arguments

##### Time

The Time argument is a string in the following format:

hh:mm:ss

where:

Time Element	Description
hh	Optional integer character string for a number of hours
mm	Optional integer character string for a number of minutes
ss	Required integer character string for a number of seconds
fff	Optional integer character string for a fraction of a second

##### WINDOW WinHandle

The optional **WINDOW** clause specifies to watch for character transmission in a specific session window. The **WINDOW** clause is composed of both the **WINDOW** keyword and the **WinHandle** argument.

The optional **WinHandle** argument is an integer that identifies a particular session window.

#### Comments

If the **WINDOW** clause is not included in the command, DCS will watch for character transmissions in the active session window.

---

## WAIT QUIET, *continued*

---

### Example

These commands:

```
CONNECT
WAIT STRING "login:"
SEND $UserName | "{ENTER}"
WAIT QUIET "00:02"
SEND $Password | "{ENTER}"
```

demonstrate an example of a simple login script for a 3270 terminal. DCS connects to the host, waits for the characters "login:" to appear, and then sends the string in \$UserName followed by a carriage return. DCS then suspends the scripts execution until communication activity has stopped for two seconds. At this point in this script segment, the host is assumed to be ready for the password.

These commands:

```
DIAL
WAIT STRING "login:"
SEND $UserName
WAIT QUIET "00:02"
SEND $password
```

demonstrate another example of a simple login script, but for a non-IBM TN3270 terminal.



---

## WAIT RESUME

---

### WAIT RESUME

The WAIT RESUME command pauses execution until you select the Resume option in the Script menu, or DCS executes a RESUME command.

### Arguments

The WAIT RESUME command takes no arguments.

### Example

In this example:

```
DIALOG "EDIT"  
EDITTEXT 120 "Enter Name:"  
BUTTON DEFAULT "OK" RESUME  
BUTTON CANCEL "CANCEL" CANCEL  
DIALOG END  
WAIT RESUME  
DIALOG CANCEL
```

DCS displays a dialog box, then waits for an option to be selected before resuming script execution.

---

# WAIT SCREEN

---

## WAIT SCREEN WINDOW WinHandle

The WAIT SCREEN command pauses script execution until the remote host changes the data in the session window.

### Arguments

#### WINDOW WinHandle

The optional **WINDOW** clause is composed of both the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular session window.

The **WINDOW** clause directs DCS to wait for data to change in a specific session window.

### Comments

Only one character needs to be updated for the session window to be considered a new screen.

If the **WINDOW** clause is not included in the command, DCS waits for data to change in the active session window.

To wait until a host is in a ready state before sending data, see the special strings for use with the SEND command.

### Example

In this example:

```
WAIT STRING "USERID"  
SEND "MYID"  
WAIT SCREEN  
SEND "PROFS"
```

DCS waits for the remote system to display the text "USERID" in the session window, then sends the string "MYID" to the remote system. After sending "MYID", DCS waits for the remote system to display new data in the session window. After the contents of the session window changes, DCS sends a command, "PROFS", to the remote system.


---

# WAIT STRING

---

## WAIT STRING String QUIET Time Window WinHandle

The WAIT STRING command pauses execution until DCS receives a string during a session.

 Note: The WAIT STRING command does not apply to the IBM TN3270 emulation.

### Arguments

#### String

The **String** argument specifies the string to wait for. The **String** argument may contain the wildcard characters ? and \*. The ? wildcard indicates that any single character may occupy that string position. The \* wildcard indicates that 0 (zero) or more characters may occupy that string position.

#### QUIET Time

The optional **QUIET** clause causes execution to resume if an amount of time elapses without character transmission, and the string is not received. The **Time** argument is a string in the following format:

Hours:Minutes:Seconds

Hours and minutes are optional, but if hours are specified, minutes must also be specified.

#### WINDOW WinHandle

The optional **WINDOW** clause directs DCS to wait for the string in a specific session window. The **WINDOW** clause is composed of both the **WINDOW** keyword and the **WinHandle** argument.

The optional **WinHandle** argument is an integer and identifies a particular session window.

### Comments

If the **WINDOW** clause is not included in the command, DCS will wait for the string in the active session window.

The **ERROR** function returns TRUE if the **QUIET** time expires.

To wait until a host is in a ready state before sending data, see the special strings for use with the **SEND** command.

### Example

In this example:

```
WAIT STRING "login" QUIET "00:00:05"
```

the script waits for the string "login" to appear in the session window. After 5 seconds, the wait string is canceled.

---

## WAIT UNTIL

---

### WAIT UNTIL Time

The WAIT UNTIL command waits until the specified time of day.

#### Arguments

##### Time

The Time argument is a string in the following format (based on a 24-hour clock):

Hours:Minutes:Seconds

Hours and minutes are optional, but if hours are specified, minutes must also be specified.

#### Example

In this example:

```
WAIT UNTIL "17:00:00"  
CONNECT
```

DCs waits until 5:00 P.M. and then connects to the host computer.

---

## WHEN CANCEL

---

### WHEN CANCEL CommandType

The WHEN CANCEL command cancels a specified WHEN command.

#### Arguments

##### CommandType

The optional **CommandType** argument consists of a command keyword and an optional command index. It specifies which WHEN command to cancel. If it is not included, all WHEN commands on the current level of execution are cancelled.

The **CommandType** argument is specified by one of the following keywords or clauses:

Keyword Clause	Action
ADVISE <b>Index</b>	Directs DCS to cancel one or more (DDE) WHEN ADVISE commands. The optional <b>Index</b> argument is an integer specifying which (DDE) WHEN ADVISE command to cancel. If it is not included, all (DDE) WHEN ADVISE commands on all levels of execution are cancelled.
COLLECT <b>Index</b>	Directs DCS to cancel one or more WHEN COLLECT commands. The optional <b>Index</b> argument is an integer specifying which WHEN COLLECT command to cancel. If it is not included, all WHEN COLLECT commands are cancelled.
DISCONNECT	Directs DCS to cancel the active WHEN DISCONNECT command.
ECHO	Directs DCS to cancel the active WHEN ECHO command.
ERROR <b>Level</b>	The ERROR clause directs DCS to cancel one or more WHEN ERROR commands. The optional <b>Level</b> argument is an integer specifying which WHEN ERROR command to cancel. If it is not included, all WHEN ERROR commands on all levels of execution are cancelled.
INITIATE	Directs DCS to cancel the active (DDE) WHEN INITIATE command.
INPUT	Directs DCS to cancel the active WHEN INPUT command.
POKE <b>Index</b>	Directs DCS to cancel one or more (DDE) WHEN POKE commands. The optional <b>Index</b> argument is an integer (from 0 to 15) specifying which (DDE) WHEN POKE command to cancel. If it is not included, all (DDE) WHEN POKE commands on all levels of execution are cancelled.
QUIET	Directs DCS to cancel the active WHEN QUIET command.
REQUEST <b>Index</b>	Directs DCS to cancel one or more (DDE) WHEN REQUEST commands. The optional <b>Index</b> argument is an integer specifying which (DDE) WHEN REQUEST command to cancel. If it is not included, all (DDE) WHEN REQUEST commands on all levels of execution are cancelled.
SCREEN <b>Index</b>	Directs DCS to cancel one or more WHEN SCREEN commands. The optional <b>Index</b> argument is an integer specifying which WHEN SCREEN command to cancel. If it is not included, all WHEN SCREEN commands on all levels of execution are cancelled.

---

## WHEN CANCEL, *continued*

---

Keyword Clause	Action
STRING <i>Index</i>	Directs DCS to cancel one or more WHEN STRING commands. The optional <i>Index</i> argument is an integer specifying which WHEN STRING command to cancel. If it is not included, all WHEN STRING commands on all levels of execution are cancelled.
TERMINATE	Directs DCS to cancel the active (DDE) WHEN TERMINATE command.
TIMER	Directs DCS to cancel the active WHEN TIMER command.
WINDOW	Directs DCS to cancel the active WHEN WINDOW command.

### Example

See the (DDE) WHEN ADVISE command.

---

## WHEN COLLECT

---

### **WHEN COLLECT Index OnStr OffStr FromStr ToStr WaitExclude Quiet limit SAVECR NOTTERMINAL WINDOW WinHandle**

The WHENCOLLECT command directs DCS to return a maximum of 254 incoming characters from a session window into a string variable until DCS receives one of the stop collecting conditions.

#### **Arguments**

##### **Index**

The **Index** argument is an integer (from 0 to 63) and is the identifier for the WHENCOLLECT command. This argument allows DCS to have a maximum of 64 WHENCOLLECT commands active at the same time. A WHENCOLLECT command replaces any previous WHENCOLLECT command that has the same **Index** argument.

##### **OnStr**

The **pOnStr** string argument directs DCS to start collecting characters when the text in the **pOnStr** is received. The collected data does not include the **pOnStr** text. Set to a null string if not used.

##### **OffStr**

The **pOffStr** string argument directs DCS to stop collecting characters when the text in the **pOffStr** is received. The collected data does not include the **pOffStr** text. Set to a null string if not used.

##### **FromStr**

The **pFromStr** string argument directs DCS to start collecting characters when the text in the **pFromStr** is received. The collected data includes the **pFromStr** text. Set to a null string if not used.

##### **ToStr**

The **pToStr** string argument directs DCS to stop collecting characters when the text in the **pToStr** is received. The collected data includes the **pToStr** text. Set to a null string if not used.

##### **ExcludeStr**

The **pExcludeStr** string directs DCS to exclude any characters contained in the **pExcludeStr** argument from the characters it collects. DCS continues to collect and exclude characters until it encounters the **pOffStr** or **pToStr** argument text or the **nQuiet** time is reached. Set to a null string if not used.

##### **Limit**

The **nLimit** integer argument directs DCS to stop collecting characters after it has received a number of characters equal to the integer in the **nLimit** argument. The **nLimit**, **nQuiet**, **pOffStr** and **pToStr** arguments can work together (DCS stops collecting characters when any condition is met). Set to a minus one (-1) if not used.

---

## WHEN COLLECT, *continued*

---

### Quiet

The `nQuiet` integer argument directs DCS to stop collecting characters after no characters are received for the `nQuiet` number of seconds. The `nQuiet`, `pOffStr` and `pToStr` arguments can work together (DCS stops collecting characters when any condition is met). Set to a minus one (-1) if not used.

### SAVECR

The `bSAVECR` argument directs DCS to store carriage returns in the collection string based on the Boolean value of the `bSAVECR` argument. If `bSAVECR` is `TRUE`, DCS saves received carriage returns. If `bSAVECR` is `FALSE`, DCS does not save received carriage returns.

### NOTERMINAL

The `bNOTERMINAL` argument if set `TRUE`, directs DCS not to pass the collected characters to the terminal emulation for processing. As the characters are not passed to the terminal emulation, they are also not displayed in the session window. If set `FALSE`, the collected characters are passes to the terminal emulation for processing.



**Note:** The `bNOTERMINAL` set `TRUE` should be used only in cases where the input to the `COLLECT` command is known. Otherwise, important terminal commands might be lost.

### WinHandle

This integer argument identifies a particular session window in DCS. If the `IWindowHandle` argument is 0 (zero), DCS uses the active session window.

### Comments

This command returns 0 (zero) if successful and 1 (one) if an error occurs.

When DCS triggers the `WhenCollect` event it passes the collected string to the corresponding void `WhenCollect(short nIndex, boolean bTimeout, BSTR pCollectBuffer, long IWindowHandle)` subroutine for processing.



---

# WHEN DISCONNECT

---

## WHEN DISCONNECT WINDOW WinHandle Command

The WHEN DISCONNECT command activates when DCS is in a wait state and when the communications connection of a session is terminated.

### Arguments

#### WINDOW WinHandle

The optional **WINDOW** clause directs DCS to check the connection for a specific session window. The **WINDOW** clause is composed of both the **WINDOW** keyword and the **WinHandle** argument.

The optional **WinHandle** argument is an integer and identifies a particular session window.

#### Command

The **Command** argument specifies a logical command (either a single command or a command block). Activation of the WHEN DISCONNECT command executes the logical command.

### Comments

If the **WINDOW** clause is not included in the command, DCS will check the connection for the active session window.

DCS is in a wait state after DCS has executed a **WAIT** command and before the condition of the **WAIT** command is fulfilled.

### Example

In this example:

```
WHEN DISCONNECT
BEGIN
DIALOG
MESSAGE "Connection terminated"
BUTTON DEFAULT "OK" RESUME
DIALOG END
WAIT RESUME
DIALOG CANCEL
QUIT
END
```

DCS displays the string "Connection terminated" in a dialog box upon disconnection.

In this example:

```
WHEN DISCONNECT
BEGIN
DISPLAY "Connection terminated"
END
```

the string "Connection terminated" displays in the session window.

---

# WHEN ECHO

---

## WHEN ECHO StringVar WINDOW WinHandle Command

The WHEN ECHO command activates when DCS is in a wait state and when DCS receives a character through the communications port for the session window.

### Arguments

#### StringVar

The StringVar argument specifies a string variable in which to store the received character.

#### WINDOW WinHandle

The optional WINDOW clause includes the WINDOW keyword and the WinHandle argument. The WinHandle argument is an integer that identifies a particular session window.

The WINDOW clause directs DCS to watch for characters in a specific session window.

### Command

The Command argument specifies a logical command (either a single command or a command block). Activation of the WHEN ECHO command executes the logical command.

### Comments

If the WINDOW clause is not included in the command, DCS watches for characters in the active session window.

DCS is in a wait state after DCS has executed a WAIT command and before the condition of the WAIT command is fulfilled.

### Example

In this example:

```
SET $char ""
WHEN ECHO $char
BEGIN
DISPLAY $char | " received"
IF $char = "Z"
RESUME
END
WAIT RESUME
```

each character that comes in through the communications port is displayed in the session window.

---

## WHEN ERROR

---

### WHEN ERROR LevelNum Command

The WHEN ERROR command activates when DCD is in a wait state and encounters an execution error during script execution.

#### Arguments

##### LevelNum

The LevelNum argument is an integer, from 0 (zero) through 3 (three), that specifies the desired error level.

##### Command

The Command argument specifies a logical command (either a single command or a command block). Activation of the WHEN ERROR command executes the logical command.

#### Comments

DCS is in a wait state after execution of a WAIT command and before the condition of the WAIT command is fulfilled. The RESULT function returns information about the error which activated the WHEN ERROR command.

#### Example

See the SET RESULT command and the TASKERROR command.

---

## WHEN INPUT

---

### **WHEN INPUT StringVar WINDOW WinHandle Command**

The **WHEN INPUT** command activates when DCS is in a wait state and when a character generated by pressing a key combination is sent out of the communications port for the session window.

#### **Arguments**

##### **StringVar**

The **StringVar** argument specifies a string variable in which to store the outgoing character.

##### **WINDOW WinHandle**

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer that identifies a particular session window.

The **WINDOW** clause directs DCS to watch for the characters entered into a specific session window.

##### **Command**

The **Command** argument specifies a logical command (either a single command or a command block). Activation of the **WHEN INPUT** command executes the logical command.

#### **Comments**

If the **WINDOW** clause is not included in the command, DCS watches for the characters entered into the active session window.

DCS is in a wait state after DCS has executed a **WAIT** command and before the condition of the **WAIT** command is fulfilled. Interface options that emulate keyboard input (such as buttons on a session toolbar) also activate the **WHEN INPUT** command. When keyboard input (real or emulated via the **SEND** command or a toolbar button) is sent out of the communications port, the **WHEN INPUT** command can be activated by each character sent. The character sent is placed into the **StringVar**, and the logical command is performed.

---

## WHEN INPUT, *continued*

---

### Example

In this example:

```
$name = "Tom"
;initializes name to "Tom"
$buffer = " "
;initializes buffer to empty
WHEN INPUT $char \
;when a character is typed,
PERFORM check_char
;calls check_char
WAIT RESUME
CANCEL

*check_char
$buffer = $buffer | $char
;concatenates character to buffer
IF ($char = "^M")
;checks for carriage return
BEGIN
IF (pos ($buffer, $name) > 0)
;beeps if "Tom" is found in a line
BEEP 1
$buffer = " "
;re-initializes buffer
END
IF ($char = "^C")
;stops buffering on [Ctrl]+[C]
RESUME
return
```

DCS calls the `check_char` routine for each character sent to the remote system. It concatenates the characters into a buffer, and when a carriage return is sent, scans the buffer for the string "Tom". If this string is found in the buffer, the computer beeps. If you press the [CTRL]+[C] key combination, the script terminates.

---

## WHEN QUIET

---

### WHEN QUIET Time WINDOW WinHandle Command

The WHEN QUIET command activates when DCS is in a wait state and has not received a character within the specified amount of time.

#### Arguments

##### Time

The Time argument is a string in the following format:

Hours:Minutes:Seconds

Hours and minutes are optional, but if hours are specified, minutes must also be specified.

##### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer and identifies a particular session window.

The **WINDOW** clause directs DCS to watch for character transmission in a specific session window.

##### Command

The **Command** argument specifies a logical command (either a single command or a command block). Activation of the WHEN QUIET command executes the logical command.

#### Comments

If the **WINDOW** clause is not included in the command, DCS watches for character transmission in the active session window.

DCS is in a wait state after execution of a **WAIT** command and before the condition of the **WAIT** command is fulfilled.

#### Example

In this example:

```
WHEN QUIET "5"  
SEND "hello"  
WAIT STRING "Ready"  
SEND "Good-bye"
```

after 5 (five) seconds pass without character transmission, DCS sends the string "hello".

---

## WHEN SCREEN

---

### WHEN SCREEN Index (Row, Col, Wid, Lns) WINDOW WinHandle Command

The WHEN SCREEN command activates when DCS is in a wait state, a region has been modified, and the cursor moves out of the modified region.

#### Arguments

##### Index

The **Index** argument is a numeric (from 0 to 23) specifying the command identifier. It allows multiple WHEN SCREEN commands to be active at the same time. A maximum of 24 WHEN SCREEN commands may exist at one time.

##### (Row, Col, Wid, Lns)

The **Row**, **Col**, **Wid**, and **Lns** arguments are integers specifying the top line, left column, column width, and optional number of lines, respectively, of the region of the terminal screen to be monitored. The first row is considered row zero. The first column is considered column 0 (zero). If the **Lns** argument is not included, the specified screen region will be 1 (one) line long.

##### WINDOW WinHandle

The **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer and identifies a particular session window.

The **WINDOW** clause directs DCS to watch the screen region in a session window specified by the **WinHandle** argument.

##### Command

The **Command** argument specifies a logical command (either a single command or a command block). Activation of the WHEN SCREEN command executes the logical command.

#### Comments

If the **WINDOW** clause is not included in the command, DCS will watch the screen region of the active session window.

DCS is in a wait state after execution of a **WAIT** command and before the condition of the **WAIT** command is fulfilled.

---

## WHEN SCREEN, *continued*

---

### Example

In this example:

```
WHEN SCREEN 0 (23, 0, 5)
INCREMENT %cnt
%cnt = 0
WHILE (%cnt < 9)
BEGIN
WAIT DELAY "2"
DISPLAY (23, 0) str (%cnt)
Display (23, 20) ""
END
```

until the variable `%cnt` equals 9, DCS displays the number of times the screen region (located at row 23, column 0 (zero), and 5 (five) characters wide) is modified.



Also see: (DDE) WHEN ADVISE command



---

# WHEN STRING

---

## WHEN STRING Index String WINDOW WinHandle Command

The WHEN STRING command activates when DCS is in a wait state and a string enters the communications port of a session window.



**Note:** The WHEN STRING command does not apply to the IBM TN3270 emulation.

### Arguments

#### Index

The optional **Index** argument is an integer (from 0 to 63) and is the identifier for the WHEN STRING command. This argument allows DCS to have a maximum of 64 WHEN STRING commands active at the same time, and allows the WHEN CANCEL command to deactivate a particular WHEN STRING command. A WHEN STRING command will replace any previous WHEN STRING command that has the same Index argument.

If you do not include the Index argument in a WHEN STRING command, that WHEN STRING command will replace all previously defined WHEN STRING commands in a script.

#### String

The String argument specifies a string.

#### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the WinHandle argument. The **WinHandle** argument is an integer and identifies a particular session window.

The **WINDOW** clause directs DCS to watch for the string in the session window specified by the **WinHandle** argument.

#### Command

The **Command** argument specifies a logical command (either a single command or a command block). Activation of the WHEN STRING command executes the logical command.

### Comments

If the **WINDOW** clause is not included in the command, DCS will watch for the string in the active session window.

DCS is in a wait state after DCS has executed a WAIT command and before the condition of the WAIT command is fulfilled.

---

## WHEN STRING, *continued*

---

### Example

This example:

```
CONNECT
WAIT STRING "Enter password"
SEND "password"
WHEN STRING "Invalid password"  CANCEL
WAIT STRING "Select topic:"
```

demonstrates a simple login procedure. DCS connects to the host and waits to receive the string "Enter password". DCS sends the password, and prepares to handle two possible responses. If the remote system indicates that the password is invalid, DCS stops execution. Otherwise, DCS resumes execution after receiving the string Select topic:.

This example:

```
DIAL "5551234"
WAIT STRING "Enter password"
SEND "password"
WHEN STRING "Invalid password"  CANCEL
WAIT STRING "Select topic:"
```

demonstrates a simple login procedure. DCS dials the specified number and waits to receive the string "Enter password". DCS sends the password, and prepares to handle two possible responses. If the remote system indicates that the password is invalid, DCS stops execution. Otherwise, DCS resumes execution after receiving the string "Select topic:".

---

## WHEN TIMER

---

### WHEN TIMER Time Command

The WHEN TIMER command activates when DCS is in a wait state and after the specified amount of time has elapsed.

#### Arguments

##### Time

The Time argument is a string in the following format:

Hours:Minutes:Seconds

Hours and minutes are optional, but if hours are specified, minutes must also be specified.

##### Command

The Command argument specifies a logical command (either a single command or a command block). Activation of the WHEN TIMER command executes the logical command.

---

## WHEN WINDOW

---

### WHEN WINDOW WinHandleVar Message WParam LParam

The WHEN WINDOW command filters out a limited subset of Windows messages to the child windows and acts upon them. This command allows for a high level of control over DCS child windows. DCS must be in a wait state before a Windows message can activate this command.



Note: The WHEN WINDOW command does not apply to IBM TN 3270 emulations.

#### Arguments

##### WinHandleVar

The WinHandleVar argument is a numeric variable that stores the handle to a particular window.

##### Message

The Message argument is a low level message which depends on actions that have occurred in Windows.

##### WParam

The WParam argument is a numeric containing information that is dependent on the particular window message.

##### LParam

The LParam argument is a numeric containing information that is dependent on the particular window message.

#### Comments

The WParam and LParam variables are assigned a value upon execution. Refer to the Microsoft Windows Software Development Kit (SDK) for further information about window messages and their corresponding parameters.

DCS is in a wait state after execution of a WAIT command and before the condition of the WAIT command is fulfilled.

#### Example

See the WINDOW DEFAULT command.

---

# WHILE

---

## WHILE Boolean Command

The **WHILE** command defines the start of a **WHILE** loop that includes a Boolean argument and a command. If the Boolean argument evaluates to **TRUE**, DCS executes the command. The Boolean argument is evaluated repeatedly and the command is executed repeatedly until the Boolean argument is **FALSE** or until DCS executes a **LEAVE** or **CONTINUE** command.

### Arguments

#### Boolean

The **Boolean** argument specifies the Boolean to be evaluated.

#### Command

The **Command** argument specifies a logical command (either a single command or a command block).

### Comments

If the **Boolean** argument is initially **FALSE**, DCS skips the command, and execution continues on the line following the command. The **Boolean** argument is evaluated before the command is executed for the first time.

### Examples

In this example:

```
SET %ndx 0
WHILE %ndx < 10
BEGIN
DISPLAY (%ndx, 0) STR (%ndx),
INCREMENT %ndx
END
```

DCS displays %ndx (converted to a string) repeatedly, until the value of %ndx equals 10 (ten).

The **WHILE** command in this example:

```
WHEN INITIATE PERFORM dde initiate
WHEN POKE 0 TABLE 0 "item0"
PERFORM poke0
WHEN TERMINATE
DISPLAY "DDE ended"
WHILE TRUE
WAIT SIGNAL
```

establishes an endless loop. After each DDE request is processed, the **WHILE TRUE** command causes script execution to wait at the (DDE) **WAIT SIGNAL** command, ready to process the next DDE request, until receipt of a Terminate message.

---

# WINDOW ACTIVATE

---

## WINDOW ACTIVATE WinHandle Boolean

The WINDOW ACTIVATE command makes a particular window the active window.

### Arguments

#### WinHandle

The WinHandle argument is a numeric expression that contains the handle of a particular window.

#### Boolean

The optional **Boolean** expression determines whether the window with handle Window will be brought into focus. If the value is TRUE (the default), the window is made the active window. If the value is FALSE, the specified window does not become the active window.

---

## WINDOW ARRANGE

---

### WINDOW ARRANGE

The WINDOW ARRANGE command tiles all open document windows so that they can be viewed at the same time.

#### Arguments

The WINDOW ARRANGE command takes no arguments.

#### Comments

Executing the WINDOW ARRANGE command is equivalent to selecting **Tile** on the **Window** menu. The document windows are arranged to allow for maximum size.

---

## WINDOW CLOSE

---

### WINDOW CLOSE WinHandle

The WINDOW CLOSE command closes a specified window and prompts you to save any changes you have made.

#### Arguments

##### WinHandle

The WinHandle argument is an integer expression specifying the handle of a particular window.

#### Comments

When using WINDOW CLOSE after a WAIT ECHO, WHEN ECHO, or WAIT PROMPT command, insert a WAIT DELAY command with a value of 1 (one) or 2 (two) following the WINDOW CLOSE command.

#### Example

This example:

```
WINDOW OPEN MEMO "" (0,0,317,245) %WIN1
WINDOW OPEN SCRIPT "Report" (316,0,317,245) True %WIN2
WINDOW OPEN SETTINGS "" FALSE %WIN3
;opens a default session window
WINDOW MOVE %WIN3 (0,247,635,150)
WAIT RESUME
WINDOW CLOSE %WIN1
WINDOW CLOSE %WIN2
WINDOW CLOSE %WIN3
```

opens three windows: a new memo, the Report script, and a new session window. The memo and script windows are positioned using arguments in the WINDOW OPEN command. The session window is positioned with the WINDOW MOVE command.



---

## WINDOW DEFAULT

---

### WINDOW DEFAULT WinHandle Message WParam LParam

The WINDOW DEFAULT command passes a message to the default handler for a window defined by the handle WinHandle. Use this command with the WHEN WINDOW command to process any window messages which your script does not process.

#### Arguments

##### WinHandle

The WinHandle argument is an integer variable that stores the handle of a particular window.

##### Message

The Message argument is a low level message which depends on actions that have occurred in Windows.

##### WParam

The WParam argument is a numeric containing information that is dependent on the particular window message.

##### LParam

The LParam argument is a numeric containing information that is dependent on the particular window message.

#### Comments

Refer to the Microsoft Windows Software Development Kit (SDK) for further information about window messages and their corresponding parameters.

#### Example

In this example:

```
WHEN WINDOW %hWnd %msg %wParam %lParam
BEGIN
WINDOW DEFAULT %hWnd %msg %wParam %lParam
IF ((%hWnd = %memoHnd) and (%msg = 0x0002))
PERFORM message_script
END
```

the WHEN WINDOW command intercepts messages for the window whose window handle is %hWnd. When a window message is interrupted by WHEN WINDOW, the message is placed in %msg. When the window message is received, the WINDOW DEFAULT command passes the window message contained in %msg to the window, and the IF command is executed. The Windows message 0x0002 specifies to destroy the memo window associated with %memoHnd.

---

## WINDOW HIDE

---

### WINDOW HIDE WinHandle

The WINDOW HIDE command hides the application window, or a child window, from view. This command is similar to using the SCREEN command with the HIDE keyword (which will hide the session window); however, the WINDOW HIDE command can hide the dcs application window or any of its child windows.

#### Arguments

##### WinHandle

The optional **WinHandle** argument is an integer and identifies a particular child window. The dcs application window has the handle number zero.

#### Comments

If you do not include the **WinHandle** argument, the command affects the active window.

If a script terminates while the DCS application window is hidden, DCS disappears from view, but still resides in memory. When DCS is in this state, it can be accessed only through another application using Dynamic Data Exchange (DDE). Without DDE, Windows regains the memory that it allocated to DCS only *after you restart Windows*.

---

# WINDOW MAXIMIZE

---

## WINDOW MAXIMIZE WinHandle

The WINDOW MAXIMIZE command enlarges the DCS window to fill the entire screen or enlarges one of the DCS child windows to fill the application window.

### Arguments

#### WinHandle

The optional **WinHandle** argument is a numeric expression that evaluates to the handle of a particular window. Use the number 0 (zero) to specify the DCS application window.

### Comments

If no window handle is specified, the command affects the active window.

Executing the WINDOW MAXIMIZE command is equivalent to selecting **Maximize** on the DCS control menu or a DCS child-window control menu.

### Example

This example:

```
WINDOW MAXIMIZE %win
```

maximizes the window specified by the window handle in the %win variable.

---

# WINDOW MESSAGE

---

## WINDOW MESSAGE WinHandle Message WParam LParam

The WINDOW MESSAGE command sends a message to the window with the handle WinHandle.

### Arguments

#### WinHandle

The WinHandle argument is a numeric expression that evaluates to the handle of a particular window.

#### Message

The Message argument is a low level message which depends on actions that have occurred in Windows.

#### WParam

The WParam argument is an integer containing information that is dependent on the particular window message.

#### LParam

The LParam argument is an integer containing information that is dependent on the particular window message.

### Comments

Refer to the Microsoft Windows Software Development Kit (SDK) for further information about window messages and their corresponding parameters.

### Example

In this example:

```
WINDOW MESSAGE %dcHnd 0x0111 0xE12A 0
;select all
WAIT DELAY '1'
WINDOW MESSAGE %dcHnd 0x0111 0xE122 0
;copy
```

all text in the window %dcHnd is selected and copied to the clipboard.

The message parameter 0x0111 corresponds to the WM\_COMMAND Windows message (see the Windows Software Development Kit for details). WM\_COMMAND is sent to a window when a menu item is chosen or a button is pressed. In this example, the WINDOW MESSAGE commands simulate selecting **Select All** (0xE12A) and **Copy** (0xE122) on the **Edit** menu.

---

## WINDOW MINIMIZE

---

### WINDOW MINIMIZE WinHandle

The WINDOW MINIMIZE command iconizes the given window, or causes DCS to run while appearing on the screen as an icon.

#### Arguments

##### WinHandle

The optional WinHandle argument is a numeric expression that evaluates to the handle of a particular window. Use the number 0 (zero) to specify the DCS application window.

#### Comments

Executing the WINDOW MINIMIZE command is equivalent to selecting Minimize on the DCS control menu.

If the window handle is not included, the command is applied to the active window.

#### Example

This example:

```
WINDOW MINIMIZE %win
```

will minimize the window specified by the window handle in the %win variable.

---

## WINDOW MOVE

---

### **WINDOW MOVE** WinHandle (x, y, w, h)

The WINDOW MOVE command moves and sizes a window.

#### **Arguments**

##### **WinHandle**

The optional **WinHandle** argument is a numeric expression that contains the handle to a particular window. The default is the DCS application window.

##### **(x, y, w, h)**

The optional coordinate set **(x, y, w, h)** specifies the desired position and size of the window. It indicates the top left corner (x, y), width (w), and height (h). If the coordinate set is included, one to four coordinates can be specified (for example, (,40,) to specify width only). If it is not included, a default size and placement is assigned.

The coordinates of child windows are relative to the position of the application window.

#### **Examples**

See the WINDOW CLOSE command.

---

## WINDOW OPEN

---

### WINDOW OPEN **Type FileName (x, y, w, h) Boolean WinHandle**

The WINDOW OPEN command opens a new empty window or a window with an existing file.

#### Arguments

##### Type

The optional **Type** argument is specified by one of the following keywords:

SETTINGS    SCRIPT    MEMO

The default window type is MEMO.

##### FileName

The **FileName** argument is a string specifying the file name to be opened. If a null string (“”) is used, a new window opens.

##### (x, y, w, h)

The optional coordinate set **(x, y, w, h)** specifies the desired position and size of the window. It indicates the top left corner (x, y), width (w), and height (h). If the coordinate set is included, one to four coordinates can be specified (for example, (,40,) to specify width only).

If this argument is not included, a default size and placement is assigned. The coordinates of the child windows are relative to the position of the application window.

##### Boolean

The optional **Boolean** argument is used to set a window in read-only or read/write mode. A TRUE state sets read-only. The default state is FALSE.

##### WinHandle

The optional **WinHandle** argument is a numeric which contains the handle to the window of the file described by the **FileName** argument.

If this argument is included but the file cannot be opened, 0 (zero) is returned. If the file does not exist or if an incorrect path is given for the file, the file cannot be opened.

#### Comments

If the maximum number of windows are open, another window is not opened. If you have included the **WinHandle** argument in the command, a 0 (zero) is placed in the argument.

The maximum number of open windows is limited only by the specific limitations of your computer, including the amount of available memory, number of communication ports, etc.

---

## WINDOW OPEN, *continued*

---

### Example

This example:

```
WINDOW OPEN MEMO "" (0,0,317,245) %WIN1
WINDOW OPEN SCRIPT "" (316,0,317,245) TRUE %WIN2
WINDOW OPEN SETTINGS "" FALSE %WIN3
WINDOW MOVE %WIN3 (0,247,635,150)
WAIT RESUME
WINDOW CLOSE %WIN1
WINDOW CLOSE %WIN2
WINDOW CLOSE %WIN3
```

opens three windows: a new memo, a new script, and a new session window. The memo and script windows are positioned using arguments in the `WINDOW OPEN` command. The session window is positioned with the `WINDOW MOVE` command.



---

# WINDOW RESTORE

---

## WINDOW RESTORE WinHandle

The WINDOW RESTORE command restores the DCS application window size and placement prior to execution of a WINDOW MAXIMIZE or WINDOW MINIMIZE command.

### Arguments

#### WinHandle

The optional **Window** argument is a numeric that evaluates to the handle of a particular window.

### Comments

Executing the WINDOW RESTORE command is equivalent to selecting **Restore** on the DCS control menu or a child window control menu.

If the window handle is not included, the command is applied to the active window.

### Example

In this example:

```
WINDOW OPEN MEMO "" (0,0,317,245) %WIN1
WINDOW MINIMIZE %WIN1
WINDOW RESTORE %WIN1
```

the memo window specified by the variable %WIN1 is opened in the upper left corner of the screen. This window is minimized, then restored to its previous size.

---

## WINDOW STACK

---

### WINDOW STACK

The WINDOW STACK command positions all open windows in the DCS application window such that the top left corner of each window is accessible.

#### Arguments

The WINDOW STACK command takes no arguments.

#### Comments

Executing the WINDOW STACK command is equivalent to selecting Cascade on the **Window** menu.

The windows are stacked from top to bottom without changing their current order.

---

## WINDOW UNHIDE

---

### WINDOW UNHIDE WinHandle

The WINDOW UNHIDE command displays the application window or a child window that the WINDOW HIDE command has hidden. The window appears with the same size and position properties prior to execution of the WINDOW HIDE command.

#### Arguments

##### WinHandle

The optional **WinHandle** argument is an integer that identifies a particular child window to be restored. The default window is the DCS application window.

---

# XFERCONFIG

---

## XFERCONFIG String WINDOW WinHandle

The XFERCONFIG command is used to set the value of a parameter used in the file transfer protocol configuration for the active session.

### Arguments

#### String

The String argument represents a single keyword followed by the assignment operator (=) and a valid setting. Together, the keyword and the setting are used to configure the file transfer protocol for the active session.



**Note:** Configuration keywords for transfer protocols shipped with client options, such as IXF and IND\$File, are valid only if the client option has been installed.

Keywords for KERMIT	Valid Setting(s)
Kermit_TimeConstraintsType	standard, loose
Kermit_ErrorCheckType	1bytechecksum, 2bytechecksum, 3bytecrc
Kermit_FileType	binary, text
Kermit_FileExist	prompt, skip, overwrite
Kermit_WindowChoice	1 (true), 0 (false)
Kermit_PacketChoice	1 (true), 0 (false)
Kermit_WindowSize	integer (1-31)
Kermit_PacketSize	integer (80-9024)
Kermit_ChangeFileName	1 (true), 0 (false)
Kermit_AutoHostActivation	1 (true), 0 (false)
Kermit_KermitCommand	command string
Kermit_KermitServerCommand	command string
Kermit_BinaryXferCommand	command string
Kermit_TextXferCommand	command string
Kermit_SendPktLenCommand	command string
Kermit_RcvPktLenCommand	command string
Kermit_WindowSizeCommand	command string
Kermit_ExitKermitCommand	command string
FT_ENABLETIMEOUT	1 (true), 0 (false)
FT_TIMEOUTVALUE	0-9999
FT_DISABLESTATUSSTOP	1 (true), 0 (false)

---

## XFERCONFIG, *continued*

---

<b>Keywords for XYMODEM</b>	<b>Valid Setting(s)</b>
XYOPTION	xmodem, ymodem, ymodemg
XYFILETYPE	binary, ascii
XYFILEEXIST	prompt, overwrite, resume
XYBLOCKSIZE	128, 1k
XYERRORCHECK	checksum, crc16
XYMAXIMUMRETRIES	Integer
XMODEM_AUTOHOSTACTIVATION	1 (true), 0 (false)
XMODEM_RCVCOMMAND	string
XMODEM_SENDCOMMAND	string
YMODEM_AUTOHOSTACTIVATION	1 (true), 0 (false)
YMODEM_RCVCOMMAND	string
YMODEM_SENDCOMMAND	String
FT_ENABLETIMEOUT	1 (true), 0 (false)
FT_TIMEOUTVALUE	0-9999
FT_DISABLESTATUSSTOP	1 (true), 0 (false)

<b>Keywords for ZMODEM</b>	<b>Valid Setting(s)</b>
ZERRORCHECK	crc16, crc32, auto
ZFILETYPE	binary, ascii
ZFILEEXIST	prompt, resume, skip, overwrite
ZAUTOSTART	1 (true), 0 (false)
ZBLOCKSIZE	128, 256, 512, 1K, auto
ZMAXIMUMRETRIES	integer
ZCONSECUTIVERETRIES	integer
ZMODEM_AUTOHOSTACTIVATION	1 (true), 0 (false)
ZMODEM_RCVCOMMAND	string
ZMODEM_SENDCOMMAND	string
ZMODEM_ESCCTRLCHAR	1 (true), 0 (false)
ZMODEM_CHANGEFILENAME	1 (true), 0 (false)
FT_ENABLETIMEOUT	1 (true), 0 (false)
FT_TIMEOUTVALUE	0-9999
FT_DISABLESTATUSSTOP	1 (true), 0 (false)

---

## XFERCONFIG, *continued*

---

If the IBM TN3270 Client Option has been installed, the following keywords are available:

Keywords for IND\$File	Valid Setting(s)
IND3270_HostEnvironment	CMS, TSO, CICS
IND3270_ChkBlockSize	1 (true), 0 (false)
IND3270_BlockSize	1 to 6233
IND3270_ChkRecLength*	1 (true), 0 (false)
IND3270_RecLength*	1 to 32760
IND3270_RecFormat	Default, Fixed, Variable, Undefined
IND3270_Append	1 (true), 0 (false)
IND3270_ConvertASCII	1 (true), 0 (false)
IND3270_ConvertCRLF	1 (true), 0 (false)
IND3270_ChkSpace**	1 (true), 0 (false)
IND3270_SpacePrimary**	0 to 99999
IND3270_SpaceSecondary**	0 to 99999
IND3270_SpaceUnits**	Blocks, Tracks, Cylinders
IND3270_Program_Name	<i>String. The string should be the name of the host program that transfers files between the host and DCS, usually IND\$File.</i>
IND3270_PacketSize	256 to 32767
IND3270_PC_CodePage	ISO Latin-1 (Windows NSI), English-US, Canadian French
IND3270_Host_CodePage	English-US, Canadian French
IND3270_Extra_Options	<i>Parameters that your host requires for which neither the Session or Advanced tabs provide settings.</i>
FT_ENABLETIMEOUT	1 (true), 0 (false)
FT_TIMEOUTVALUE	0-9999
FT_DISABLESTATUSSTOP	1 (true), 0 (false)

\* Applicable only to CMS and TSO environments.

\*\* Applicable only to TSO environments.

---

## XFERCONFIG, *continued*

---

If the Tandem 6530 Client Option is installed, the following keywords are available:

Keywords for IXF	Valid Setting(s)_
IXF_HOSTPROGRAM	string (48 character limit)
IXF_MAXRETRIES	positive integer (5 digit limit)
IXF_DEFAULTVOL	volume/subvolume name
IXF_RECORDSIZE	positive integer (1 – 4096)
IXF_SHARE	1 (true), 0 (false)
IXF_PACKETDEPTH	1 – 16
IXF_BINARY	true (Binary transfer), false (Text transfer)
IXF_FILECODE	0 – 65535 (may not be 101 for binary transfer)
IXF_EDITINCREMENT	0.001 – 2000.999
IXF_ENABLEPRINTER	1 (true), 0 (false)
IXF_STRIPHIGHBIT	1 (true), 0 (false)
IXF_SKIPPERFS	1 (true), 0 (false)
IXF_MAKEEXTENSION	1 (true), 0 (false)
IXF_NUMEXTENSIONCHARS	1 – 3
IXF_CRLF	1 (true), 0 (false)
IXF_KEEPCFILEDATE	1 (true), 0 (false)
IXF_RCVFILEEXISTOPTION	ReceiveAbort, ReceivePrompt, ReceiveOverwrite, ReceiveAppend
IXF_NOEXTENSIONS	1 (true), 0 (false)
IXF_OWNER	1 (true), 0 (false)
IXF_GROUP	0 – 255
IXF_USER	0 – 255
IXF_FILESECURITY	four character string, using A, N, G, O, C, U, S (may also be -), or D
IXF_PRIMARYHOSTTEXT	1 – 65535
IXF_SECONDARYHOSTTEXT	1 – 65535
IXF_KEEPPCFILEDATE	1 (true), 0 (false)
IXF_SENDFILEEXISTOPTION	SendAbort, SendPurge, SendErase, SendAppend
IXF_TABOPTION	DeleteTabs, TabsToSpaces, TabList
IXF_TABSEVERY	positive integer (1 – 4096)
IXF_TABLIST	comma-delimited list of up to 128 Tab Stops in range 1 – 4096
FT_ENABLETIMEOUT	1 (true), 0 (false)
FT_TIMEOUTVALUE	0-9999
FT_DISABLESTATUSSTOP	1 (true), 0 (false)

---

---

## XFERCONFIG, *continued*

---

### WINDOW WinHandle

The optional **WINDOW** clause includes the **WINDOW** keyword and the **WinHandle** argument. The **WinHandle** argument is an integer identifying a particular child window.

The **WINDOW** clause applies the file transfer configuration settings to a particular session window.

### Comments

The keywords correspond to parameters available on the **File Transfers** tab of the **Session Properties** dialog. This command, therefore, allows you to set these parameters through the DCS Script Language rather than using the dialog box.

The **ERROR** function returns **TRUE** if the **WinHandle** or **String** keyword is invalid.

If **WinHandle** is not specified, the file transfer protocol configuration settings are applied to the active session window.

### Example

In this example:

```
XFERCONFIG "BlockSize=256"
```

the block size for the transfer is set to 256 bytes.



A

---

*Task Errors*

DCS

---

## Task Errors

---

When an error occurs during script execution, DCS displays a four digit number in an error dialog. Here's an example:

3411

### First Digit

The first digit in the error number is the level at which the error occurred. The following table lists all error levels with a short description of each:

Error Level	Description
0	Fatal
1	Critical
2	Warning
3	Run Time

### Remaining Digits

The three digits following the error level indicate the actual error number. The following table lists all three digit error numbers with a short description of each.



Also see: `TASKERROR` command

---

## Task Errors, *continued*

---

Error #	Cause of the Error
101	Occurs if a memory allocation or a memory lock fails.
111	Occurs when an attempt is made to assign a value to a variable and it fails.
112	Occurs when a RECORD FORMAT command is executed with illegal row/column parameters, or a memory allocation in which to store the information fails.
113	Occurs when trying to add a dialog control and it cannot add the new control information to the master dialog control structure. This is due to a memory shortage.
114	Occurs if an unsuccessful attempt to save a keymap file has been made.
115	The keyname parameter resolves to an integer less than zero or greater than 255 when using the KEY command.
201	Occurs when trying to execute a FILE command and the source file does not exist.
202	Occurs when using a FILE RENAME command and the destination file already exists.
211	Occurs if an unsuccessful attempt to launch another application has been made.
301	No terminal window is open when using the following commands: DIAL, BREAK, DROPDTR, FILE SEND..., FILE RECEIVE..., SENDBREAK, SETTINGS, SAVE, and any SCREEN... command (except SCREEN HIDE).
321	The text specified by the EDIT FIND or EDIT REPLACE commands is not found.
331	The window specified by the WINDOW OPEN command was not opened for any reason.
341	The SET ATTRIBUTES command fails to set the specified attributes.
351	When using the FILE OPENNAME or FILE CREATENAME commands and the user cancels the dialog without choosing a file name.
361	Occurs when using the LIBRARY LOAD command and the maximum number DLLs the script may load has been exceeded, or if the command returns an error value.
362	Occurs when using the LIBRARY CALL command and GetProcAddress cannot resolve the entry point of the specified DLL function or GlobalReAlloc fails in allocating space to pass parameters from a table.
363	Occurs when using the LIBRARY UNLOAD command returns an error value.
401	Occurs when trying to decode the next command and it is not a valid command.
402	Occurs if a valid command is followed by an option that the command does not use.
403	Unused; ignore if displayed.
404	Occurs if trying to use TRANSFER OBJECT and the application is not running under NewWave.
408	Occurs if an attempt is made to divide by zero in the expression.

---

## Task Errors, *continued*

Error #	Cause of the Error
411	Occurs when an integer variable is needed but not present, or when variable is present but the wrong type. Cases: FILE OPENNAME or FILE CREATENAME WAIT COLLECT WHEN INPUT WHEN ECHO (DDE) ACCESS (with the optional DDEList) (DDE) WHEN REQUEST WHEN EXECUTE CONCAT (obsolete, no longer documented, but kept for backwards compatibility) PARSE
412	Occurs when an integer variable is needed but not present, or when variable is present but the wrong type. Cases: LIBRARY LOAD library handle (LibraryName) WINDOW OPEN window handle (WinHandle) WHEN WINDOW any of the four variables (WinHandleVar, Message, WinParam, LevelParam) (DDE) ACCESS channel variable (ChannelVar) (DDE) WHEN table number (TABLE Table) INCREMENT or DECREMENT variable (IntVar)
421	Occurs when a string expression is expected but not found.
422	Occurs when an integer expression is expected but not found.
423	Occurs when a Boolean expression is expected but not found.
424	Occurs when a branch can not be resolved. Cases: EXECUTE (a jump using a string denoted target name that has the string missing) GOTO (same as EXECUTE) PERFORM (same as EXECUTE)
425	Occurs when it expects a valid time expression. Cases: WAIT DELAY WAIT UNTIL WAIT QUIET WAIT STRING WHEN TIMER WHEN QUIET
426	Occurs when expecting (x, y, width, height) and there is an error.
501	Occurs when trying to execute a WHEN xxxx Number. Where xxxx is STRING, POKE, etc. and Number exceeds the maximum number of WHEN commands for the particular case of xxxx.

## Task Errors, *continued*

Error #	Cause of the Error
502	Occurs when trying to setup a WHEN SCREEN command and the user tries to define an area that is not on the screen.
511	No longer used.
524	NetBios error.
525	Connection already made.
526	Already connected to host.
601	Occurs when trying to use a table that has not been defined or when creating a list box in a dialog that uses an undefined table in a DDE transaction Cases: TABLE SORT
602	Occurs when a table number less than zero or greater than MAXTABLE- NUM-1 Cases: Any TABLE command any RECORD command updating a dialog list box using an undefined table in a DDE transaction using an undefined table in a (DDE) WHEN command
603	Invalid field count.
604	A table to be sorted with the SORT command contains zero records.
605	Occurs when using the TABLE LOAD command and the source data fails to load for any reason.
606	Occurs when using the TABLE SAVE command and the data fails to be writ- ten to the destination specified for any reason
611	Occurs when trying to update a list box and an invalid record number is given.
621	Occurs when using the TABLE COPY command and the source and desti- nation tables are the same.
651	Occurs if you attempt to add popups or menu items without first executing the MENU command.
661	Occurs if you try to add an item and there are no popups defined.
671	Occurs when trying to access undefined menu or item. Cases: Try to access an undefined system menu Try to access an undefined user menu Try to access an undefined item in the system menu Try to access an undefined item in a user menu
701	Try to create or update an item in a dialog without having executed the DIA- LOG command.
702	Occurs if you try to have a total of more than 255 dialog controls.
711	Occurs if you try to define more than MAX_WHEN_BUTTONS buttons. This includes buttons, wide buttons, and icon buttons.
712	Occurs if you try to define more than MAX_WHEN_GROUPS radio groups.
713	Occurs if you try to use a radio button without a radio group.
714	Occurs if you try to define more than MAX_WHEN_CHECKS check boxes.
715	Occurs if you try to define more than MAX_DLG_EDITS edit texts.
716	Occurs if you try to define more than MAX_WHEN_LISTS list boxes.

## Task Errors, *continued*

Error #	Cause of the Error
721	Occurs if you try to use a LIMIT qualifier in a (DIALOG) EDITTEXT command with a value greater than STR255-2 ( 256-2 ).
731	Occurs if you try to perform a DIALOG UPDATE on a nonexistent item.
732	Occurs when the specified ICON resource does not exist.
801-816	The WAIT errors occur when memory cannot be allocated in which to store the WAIT information.
820-844	The WHENENABLE errors occur when memory cannot be allocated to store the WHEN lookup information.
851-874	The WHENACTIVATE errors occur when memory cannot be allocated to store the WHEN activation information.
901	Occurs if memory cannot be allocated to store information about the current state of execution before a branch to a subroutine.
902	Occurs if a RETURN command cannot be executed correctly.
903	Occurs if PERFORM or EXECUTE fails.
904	Occurs if the system cannot halt the current level of operation. Most likely to occur immediately before a branch when the system is trying to save or flush data associated with the current level of operation.
921	Could not execute a WHEN ERROR branch. Could not execute a WHEN branch.
931	Occurs if an encrypted string overflows the script string size limit of 254 bytes.
981	Occurs when using the (DDE) ACCESS command and the WM_DDE_INITIATE message fails to initiate a DDE conversation.
982	Occurs when using the (DDE) TABLE REQUEST command and the DDE server does not pass data back in CF_TEXT format.
983	Occurs when using the (DDE) TABLE POKE command and the target table is not in the CF_TEXT format.
984	Occurs when using the (DDE) TABLE REPLY command and the replying table is not in CF_TEXT format.
985	Occurs when a DDE command times out.
986	Occurs when the DDE POKE command fails.
991	Occurs if no DDE channels are currently open and a DDE transaction is attempted. Also occurs if upon failure of a DDE command to allocate or lock memory.
996	Occurs when the maximum number of WHEN ADVISE commands have been exceeded and the task engine cannot reply to an advise transaction.
997	Occurs if another DDE transaction is received before the response to a prior transaction has been acknowledged. Also occurs if the value to be decrypted is not a sting variable.
998	GlobalAlloc fails in allocating the global memory object to hold the string to be passed to the server.  If rudimentary sanity check of decrypted string fails or if incorrect password used to decrypt the string.  If the window handle passed to the WINDOW CLOSE or WINDOW SAVE commands is not a DCS child window handle.
999	Error with NetBios. Most likely it is not loaded.

# B

---

*New and Removed  
Commands and Functions*

DCS

---

## New Commands and Functions

---

The following have been added to the DCS Script Language:

### —Commands—

APPCONFIG  
CONNCONFIG  
DISPLAYCONFIG  
GENERALCONFIG  
EMULCONFIG  
KEYMAP LOAD  
KEYMAP SAVE  
LOGOTFILE  
MENU DELETE ITEM  
MENU DELETE POPUp  
MENU INSERT ITEM  
MENU INSERT POPUP  
SCROLL DOWN  
SCROLL LEFT  
SCROLL RIGHT  
SCROLL UP  
SET AUTOSCROLLTOCURSOR  
SET DEFAULTSESSIONHANDLE  
SET DDETIMEOUT  
SPAWN  
TOOLBARHIDE  
TOOLBARSHOW  
XFERCONFIG

### —Functions—

BUFFER  
DEFAULTSESSIONHANDLE  
DIALOGHANDLE  
EXPLDATTR  
FLDATTREXPOS  
FLDNUM  
FLDTEXT  
GETAPPCONFIG  
GETCONNCONFIG  
GETDISPLAYCONFIG  
GETEMULCONFIG  
GETGENERALCONFIG  
GETXFERCONFIG  
MESSAGEBOX  
SEARCHINRECT  
TYPEDLIBRARYCALL  
WNDFILE



---

## Removed Commands and Functions

---

The following have been removed from the DCS Script Language:

### —Commands—

(DIALOG) WIDEBUTTON	SET NETWORK RECEIVETIMEOUT
FILE VIEW GIF	SET NETWORK REMOTENAME
FILE VIEW RLE	SET NETWORK SENDTIMEOUT
FILE VIEW TEXT	SET PARITYCHECK
KEYCODE	SET PINTER
MERGE	SET RECORDFORMAT
RESET SERIAL	SET RECORDLENGTH
SET ANSWERBACK	SET SCROLLMARGINS
SET AUTORESET	SET SPACE1
SET BLOCKSIZE	SET SPACE2
SET CONFIRMDISCONNECT	SET STOPALERT
SET CONNECTOR	SET TASKYIELD
SET COUNTRY	SET TERMSCROLLBARS
SET CURSORBLINK	SET TERMTITLE
SET EMULATE	SET TEXTTRANSFERS
SET FKEYSARRANGE	SET TIMEOUT
SET FKEYSICONS	SET TRANSLATION
SET FKEYSONEROW	SET TRUEATTRIBUTES
SET FOLDER	SET UNITS
SET IBMTOANSI	TIMER
SET INCOMINGCR	VIDEO LOAD
SET INCOMINGLF	VIDEO RESET
SET INPUTMODE	VIDEO SAVE
SET LIMITWINDOWSIZE	VIDEO STYLE
SET LINEWRAP	VIDEO UPDATE
SET MEMOTITLE	WAIT NETWORK
SET MODEM	WAITFORCALL
SET MODEMTYPE	WHEN LISTEN
SET MONITORMODE	WHEN RECEIVE
SYSTEM 0x801 (use MESSAGEBOX() function)	WINDOW ATTACHMENT
SET NETWORK	WINDOW SAVE
SET NETWORK CARDNUM	WINDOW SELECT
SET NETWORK LINK	WINDOW SHOW
SET NETWORK LOCAL NAME	
SET NETWORK LUNUM	
SET NETWORK LUPUNAME	

---

## Removed Commands and Functions, *continued*

---

### —Functions—

BUFFER ( )

FOLDER ( )

HWNDFILE ( )

KEYBOARDSTATE ( )

MODEMTYPE

SELECTION ( )

SETTINGS (Country)

SETTINGS (Translation)

VIDEO ( )

# C

---

*Quick Reference for  
Command and Function Syntax*

DCS

## Syntax Quick Reference

All functions and commands available in the DCS Script Language are listed below with the required statement syntax for each. Functions are followed by “( )” characters.



**Note:** Commands or functions flagged with an asterisk do not apply to the IBM TN3270 emulation.

Command/Function Name	Statement Syntax
<b>–A–</b>	
ACTIVE ( )	ACTIVE ( )
APPCONFIG	APPCONFIG String
ARGUMENTS	ARGUMENTS (Parameter, ...)
ATTRIBUTES ( )	ATTRIBUTES (FileName)
<b>–B–</b>	
BAND ( )	BAND (Source, Mask)
BEEP*	BEEP Num
BEGIN	BEGIN
BNOT ( )	BNOT (Numeric)
BOOL ( )	BOOL (Integer)
BOR ( )	BOR (Source, Mask)
BREAK*	BREAK DelayUnits
BXOR ( )	BXOR (Source, Mask)
<b>–C–</b>	
CANCEL	CANCEL
CHR ( )	CHR (Numeric)
CLEAR	CLEAR Screen WINDOW WinHandle
COLLECT*	COLLECT UNTIL String1 EXCLUDE String2 String3 LIMIT Num SAVECR SBoolean NOTERMINAL WINDOW WinHandle
COMPILE	COMPILE Script Make Display
CONCAT	CONCAT StringVar String String ...
CONNCONFIG	CONNCONFIG String WINDOW WinHandle
CONNECT ( )	CONNECT ( )
CONNECT	CONNECT SessionFile WINDOW WinHandleVar
CONNECTMESSAGE ( )*	CONNECTMESSAGE ( )
CONNECTRESULT ( )*	CONNECTRESULT (WinHandle)
CONTINUE	CONTINUE
CREATE DIRECTORY*	CREATE DIRECTORY Path

## Syntax Quick Reference, *continued*

Command/Function Name	Statement Syntax
CURSOR ( )	CURSOR (WinHandle)
–D–	
DATE ( )	DATE (Seconds)
(DDE) ACCESS	ACCESS Server Topic ChannelVar DDEList
(DDE) ACCESS CANCEL	ACCESS CANCEL Channel
(DDE) ADVISE ( )	ADVISE ( )
(DDE) INSTRUCT	INSTRUCT Channel Command, ...
(DDE) POKE	POKE Data TO Channel Item
(DDE) REQUEST	REQUEST DataVar FROM Channel Item
(DDE) TABLE REPLY	TABLE REPLY StrucTableNum TO Channel Item
(DDE) TABLE REQUEST	TABLE REQUEST StrucTableNum FROM Channel Item AS Format
(DDE) TABLE SEND	TABLE SEND StrucTableNum TO Channel Item AS Format
(DDE) WAIT SIGNAL	WAIT SIGNAL
(DDE) WHEN ADVISE	WHEN ADVISE Index Channel Item Command
(DDE) WHEN EXECUTE	WHEN EXECUTE Channel CommandVar Command
(DDE) WHEN INITIATE	WHEN INITIATE ChannelVar Command
(DDE) WHEN POKE	WHEN POKE Index TABLE Table Channel Item Command
(DDE) WHEN REQUEST	WHEN REQUEST Index Channel Item Command
(DDE) WHEN TERMINATE	WHEN TERMINATE Channel Command
DEBUG	DEBUG FileName
DECREMENT	DECREMENT IntVar
DECRYPT ( )	DECRYPT (EncryptedString, Key)
DEFAULTSESSIONHANDLE ( )	DEFAULTSESSIONHANDLE ( )
DIAL*	DIAL PhoneNum RETRY Retries DELAY RetryDelay
DIALOG	DIALOG (x, y, w, h) Title SYSMENU SBoolean MODAL MBoolean NOFOCUS FBoolean
(DIALOG) BUTTON	BUTTON (x, y, w, h) Default Title Command
DIALOG CANCEL	DIALOG CANCEL DialogIndex
(DIALOG) CHECKBOX ( )	CHECKBOX (ControlNum, DialogIndex)
(DIALOG) CHECKBOX	CHECKBOX (x, y, w, h) Default CheckBoxText Command
DIALOG CONTROL*	DIALOG CONTROL DialogIndex Control ControlNum Update
(DIALOG) DIMENSION	DIMENSION Control Attribute Value

---

## Syntax Quick Reference, *continued*

---

Command/Function Name	Statement Syntax
(DIALOG) EDITTEXT ( )	EDITTEXT (ControlNum, DialogIndex)
(DIALOG) EDITTEXT	EDITTEXT (x, y, w, h) Box Prompt Text LIMIT NumChars PASSWORD
(DIALOG) GROUPBOX	GROUPBOX (x, y, w, h) Message
(DIALOG) ICON	ICON (x, y, w, h) IconId
(DIALOG) ICONBUTTON	ICONBUTTON (x, y, w, h) IconId Default Title Command
(DIALOG) LISTBOX ( )	LISTBOX (ControlNum, DialogIndex)
(DIALOG) LISTBOX	LISTBOX (x, y, w, h) Table Record INVERT COMBOBOX Command
(DIALOG) MESSAGE	MESSAGE (x, y, w, h) Text
(DIALOG) MESSAGEBOX ( )	MESSAGEBOX (Message, Title, Style)
(DIALOG) NEWLINE	NEWLINE
(DIALOG) PICTURE	PICTURE (x, y, w, h) PictureId
(DIALOG) RADIOBUTTON	RADIOBUTTON (x, y, w, h) ButtonName
(DIALOG) RADIOGROUP ( )	RADIOGROUP (ControlNum, DialogIndex)
(DIALOG) RADIOGROUP	RADIOGROUP (x, y, w, h) Default GroupName Command
DIALOG UPDATE	DIALOG UPDATE DialogIndex Control ControlNum Update
DIALOGHANDLE ( )	DIALOGHANDLE (Index)
DIRECTORY ( )	DIRECTORY (Type)
DISCONNECT	DISCONNECT WINDOW WinHandle
DISKSPACE ( )	DISKSPACE (Drive)
DISPLAY	DISPLAY (Row, Col) String CRONLY WINDOW WinHandle
DISPLAYCONFIG	DISPLAYCONFIG String WINDOW WinHandle
DROPDTR*	DROPDTR DelayUnits
-E-	
EDIT COPY	EDIT COPY String WINDOW WinHandle
EDIT COPYSPECIAL*	EDIT COPYSPECIAL Destination Format WINDOW WinHandle
EDIT CUT	EDIT CUT WINDOW WinHandle
EDIT FIND	EDIT FIND String CASE REVERSE WINDOW WinHandle
EDIT GOTO	EDIT GOTO Line WINDOW WinHandle

---

## Syntax Quick Reference, *continued*

---

Command/Function Name	Statement Syntax
EDIT PASTE	EDIT PASTE <b>WINDOW</b> WinHandle
EDIT REPLACE	EDIT REPLACE String1 String2 <b>CASE REVERSE</b> <b>WINDOW</b> WinHandle
EMULCONFIG	EMULCONFIG String <b>WINDOW</b> WinHandle
ENCRYPT ( )	ENCRYPT (OrigString, Key)
END	END
EOF ( )	EOF ( )
ERROR ( )	ERROR ( )
EXECUTE	EXECUTE Target
EXFLDATTR ( )	EXFLDATTR (FieldNum, WinHandle)
EXISTS ( )	EXISTS (FileName)
<b>–F–</b>	
FILE CLOSE*	FILE CLOSE <b>WINDOW</b> WinHandle
FILE COMPRESS	FILE COMPRESS FileName
FILE COPY	FILE COPY Source TO Destination <b>APPEND</b>
FILE CREATENAME	FILE CREATENAME FileName <b>TYPE</b> Type <b>DEFAULT</b> <b>String</b> <b>TITLE</b> TitleText <b>NOCREATE</b>
FILE DECOMPRESS	FILE DECOMPRESS FileName
FILE DECRYPT	FILE DECRYPT FileName Key
FILE DELETE	FILE DELETE FileName
FILE ENCRYPT	FILE ENCRYPT FileName Key
FILE OPENNAME	FILE OPENNAME FileName <b>TYPE</b> Type <b>DEFAULT</b> <b>String</b> <b>TITLE</b> TitleText
FILE PAUSE*	FILE PAUSE
FILE RECEIVE BINARY	FILE RECEIVE <b>REMOTE</b> BINARY FileName <b>AS</b> Dest <b>ASCII</b> <b>BINARY</b> <b>CRLF</b> <b>NOCLRF</b> <b>NEITHER</b> <b>APPEND</b> <b>WINDOW</b> WinHandle
FILE RENAME	FILE RENAME Source TO Destination
FILE RESUME*	FILE RESUME <b>WINDOW</b> WinHandle
FILE SEND BINARY	FILE SEND <b>REMOTE</b> BINARY FileName <b>AS</b> Hostname <b>ASCII</b> <b>BINARY</b> <b>CRLF</b> <b>NOCLRF</b> <b>APPEND</b> <b>WINDOW</b> WinHandle
FILESIZE ( )	FILESIZE (FileName)
FILTER ( )	FILTER (String, SearchChars, ReplaceChars)
FKEYS	FKEYS Display
FLDATTR ( )	FLDATTR (Field, WinHandle)

---

## Syntax Quick Reference, *continued*

---

Command/Function Name	Statement Syntax
FLDATTREXPOS ( )	FLDATTREXPOS (Row, Column, WinHandle)
FLDLEN ( )	FLDLEN (Field, WinHandle)
FLDNUM ( )	FLDNUM (Row, Column, WinHandle)
FLDPOS ( )	FLDPOS (Field, WinHandle)
<b>–G–</b>	
GENERALCONFIG	GENERALCONFIG String WINDOW WinHandle
GETAPPCONFIG ( )	GETAPPCONFIG (KeyString)
GETCONNCONFIG ( )	GETCONNCONFIG (KeyString, WinHandle)
GETDISPLAYCONFIG ( )	GETDISPLAYCONFIG (KeyString, WinHandle)
GETEMULCONFIG ( )	GETEMULCONFIG (KeyString, WinHandle)
GETGENERALCONFIG ( )	GETGENERALCONFIG(KeyString, WinHandle)
GETPROFILEDATA ( )	GETPROFILEDATA (Section, KeyName, INIFile)
GETXFERCONFIG ( )	GETXFERCONFIG (KeyString, WinHandle)
GOTO	GOTO Target
<b>–H–</b>	
HANGUP*	HANGUP WINDOW WinHandle
HWNDLIST ( )	HWNDLIST (NumWin)
<b>–I–</b>	
ICONIC ( )	ICONIC (WinHandle)
IF	IF Boolean Command ELSE Command
INCREMENT	INCREMENT IntVar
INT ( )	INT (RealNum)
<b>–K–</b>	
KERMIT COPY*	KERMIT COPY SourceFile TO DestinationFile WINDOW WinHandle
KERMIT DIRECTORY*	KERMIT DIRECTORY DirectoryName WINDOW WinHandle
KERMIT ERASE*	KERMIT ERASE FileName WINDOW WinHandle
KERMIT FINISH*	KERMIT FINISH WINDOW WinHandle
KERMIT FREESPACE*	KERMIT FREESPACE WINDOW WinHandle
KERMIT HELP*	KERMIT HELP HelpTopic WINDOW WinHandle
KERMIT LOGOUT*	KERMIT LOGOUT WINDOW WinHandle



---

## Syntax Quick Reference, *continued*

---

Command/Function Name	Statement Syntax
KERMIT MESSAGE*	KERMIT MESSAGE ReceiverName MessageText <b>WINDOW WinHandle</b>
KERMIT NEWDIRECTORY*	KERMIT NEWDIRECTORY DirectoryName <b>WINDOW WinHandle</b>
KERMIT RENAME*	KERMIT RENAME OldFileName TO NewFileName <b>WINDOW WinHandle</b>
KERMIT TYPE*	KERMIT TYPE FileName <b>WINDOW WinHandle</b>
KERMIT WHO*	KERMIT WHO UserName <b>WINDOW WinHandle</b>
KEY	KEY Modifier1 Key1 Definition <b>WINDOW WinHandle</b>
KEYBOARD	KEYBOARD State <b>WINDOW WinHandle</b>
KEYMAP LOAD	KEYMAP LOAD FileName
KEYMAP RESET	KEYMAP RESET
KEYMAP SAVE	KEYMAP SAVE FileName
-L-	
LAUNCH	LAUNCH Application Command, ... ContinueScript
LEAVE	LEAVE
LENGTH ( )	LENGTH (String)
LEVEL	LEVEL KeyLevel
LIBRARY CALL	LIBRARY CALL LibraryReference Procedure (Parameter, ...)
LIBRARY LOAD	LIBRARY LOAD LibraryName LibraryReference
LIBRARY UNLOAD	LIBRARY UNLOAD LibraryReference
LINENUMBERS	LINENUMBERS
LOAD	LOAD FileName
LOGTOFILE	LOGTOFILE FileName <b>WINDOW WinHandle</b>
-M-	
MENU	MENU
MENU CANCEL	MENU CANCEL
(MENU) CHECKED ( )	CHECKED (Popup, Item)
(MENU) DELETE ITEM	MENU DELETE ITEM IntPopup IntItem
(MENU) DELETE POPUP	MENU DELETE POPUP IntPopup
(MENU) ENABLED ( )	ENABLED (Popup, Item)
(MENU) INSERT ITEM	MENU INSERT ITEM IntPopup IntItem StrText Enabled Checked Command

---

## Syntax Quick Reference, *continued*

---

Command/Function Name	Statement Syntax
(MENU) INSERT POPUP	MENU INSERT POPUP IntPopup StrText
(MENU) ITEM	ITEM Text <b>Enabled</b> <b>Checked</b> Command, ...
(MENU) POPUP	POPUP Text <b>SYSTEM</b> Pos
(MENU) SEPARATOR	SEPARATOR
MENU UPDATE	MENU UPDATE Popup Item Text <b>Enabled</b> <b>Checked</b>
<b>–N–</b>	
NETID (*)	NETID ( )
NEXT ( )	NEXT ( )
NOSHOW	NOSHOW
NUM ( )	NUM (String)
<b>–O–</b>	
ORD ( )	ORD (String)
<b>–P–</b>	
PARSE	PARSE String StringVar1 Keyword StringVar2
PASSWORD (*)	PASSWORD ( )
PERFORM	PERFORM Target ( <b>Parameter</b> , ...)
PHONENUMBER (*)	PHONENUMBER ( )
POS ( )	POS (String, Keyword, <b>Start</b> )
POSITION ( )	POSITION (WinHandle, <b>Boolean</b> )
POWER ( )	POWER (Base, Exponent)
PRINT CANCEL	PRINT CANCEL
PRINT CLOSE	PRINT CLOSE
PRINT FILE	PRINT FILE FileName LF
PRINT FONT	PRINT FONT Font Point
PRINT NEWLINE	PRINT NEWLINE
PRINT NEWPAGE	PRINT NEWPAGE
PRINT OPEN	PRINT OPEN PORT Port <b>DRIVER</b> Driver <b>TYPE</b> Type <b>ABORT</b>
PRINT STRING	PRINT STRING String
PRINT STYLE	PRINT STYLE Attribute ...
PRINT TABS	PRINT TABS Width
PRINT TERMINAL*	PRINT TERMINAL State <b>WINDOW</b> WinHandle
PRTMETRICS ( )	PRTMETRICS ( )

---

## Syntax Quick Reference, *continued*

---

Command/Function Name	Statement Syntax
PUTPROFILEDATA ( )	PUTPROFILEDATA (Section, KeyName, Data, File)
-Q-	
QUIT	QUIT
-R-	
RANDOM ( )	RANDOM (Range)
REAL ( )	REAL (Numeric)
RECORD FORMAT	RECORD FORMAT Lines (Row1 Col1) ... (Rown Coln) WINDOW WinHandle fields
RECORD READ	RECORD READ Table AT Record AT Position LENGTH Bytes
RECORD SCAN	RECORD SCAN Table WINDOW WinHandle
RECORD WRITE	RECORD WRITE Table AT Record AT Position LENGTH Bytes
REMOVE DIRECTORY*	REMOVE DIRECTORY Path
RESETSERIAL	RESETSERIAL
RESTART	RESTART
RESULT ( )	RESULT ( )
RESUME	RESUME
RETURN	RETURN
ROUND ( )	ROUND (Real, Places, Boolean)
ROUTE ( )	ROUTE (FileSpec, ATTRIBUTES Type)
-S-	
SAVE	SAVE FileName
SCREEN ( )	SCREEN (Row, Column, Length, WinHandle)
SCREEN	SCREEN (x, y, w, h) Display WINDOW WinHandle
SCROLL DOWN	SCROLL DOWN Rows
SCROLL LEFT	SCROLL LEFT Columns
SCROLL RIGHT	SCROLL RIGHT Columns
SCROLL UP	SCROLL UP Rows
SEARCH ( )	SEARCH (Row, Column, Length, String, WinHandle)
SEARCHINRECT ( )	SEARCHINRECT (TopRow, BottomRow, LeftCol, RightCol, String, WinHandle)
SECONDS ( )	SECONDS (Time, Date)
SELECTION	SELECTION StartLine EndLine WINDOW WinHandle

---

## Syntax Quick Reference, *continued*

---

Command/Function Name	Statement Syntax
SELECTION APPEND	SELECTION APPEND FileName <b>TABLE WINDOW WinHandle</b>
SELECTION BUFFER	SELECTION BUFFER <b>WINDOW WinHandle</b>
SELECTION PRINT	SELECTION PRINT <b>WINDOW WinHandle</b>
SELECTION SAVE	SELECTION SAVE FileName <b>TABLE WINDOW WinHandle</b>
SELECTION SEND	SELECTION SEND <b>WINDOW WinHandle</b>
SEND	SEND (Row, Col) NOCR String <b>ONEPACKET WAITECHO WINDOW WinHandle</b>
SEENDBREAK*	SEENDBREAK DelayUnits
SET	SET Variable Source
SET APPTITLE	SET APPTITLE Name
SET ATTRIBUTES	SET ATTRIBUTES FileName Attribute
SET AUTOSCROLLTOCURSOR	SET AUTOSCROLLTOCURSOR Boolean
SET AUTOSIZE*	SET AUTOSIZE Boolean
SET BACKSPACEDESTRUCTIVE*	SET BACKSPACEDESTRUCTIVE Boolean
SET BACKSPACEKEY*	SET BACKSPACEKEY Keyword
SET BAUDRATE*	SET BAUDRATE Rate
SET BINARYTRANSFERPARAMS*	SET BINARYTRANSFERPARAMS OptionKeyWord ActionKeyWord OptionKeyWord ActionKeyWord...
SET BINARYTRANSFERS	SET BINARYTRANSFERS Protocol <b>WINDOW WinHandle</b>
SET BUFFERLINES	SET BUFFERLINES Lines
SET CARRIERDETECT*	SET CARRIERDETECT Boolean
SET COLUMNS	SET COLUMNS Integer
SET CONNECTION*	SET CONNECTION Connector <b>Command WINDOW WinHandle</b>
SET CONNECTMESSAGE*	SET CONNECTMESSAGE Text
SET CONNECTRESULT*	SET CONNECTRESULT Num
SET CURSOR	SET CURSOR Display
SET DATABITS*	SET DATABITS Integer
SET DDETIMEOUT	SET DDETIMEOUT <b>Seconds</b>
SET DECIMAL	SET DECIMAL Numeric
SET DEFAULTSESSIONHANDLE	SET DEFAULTSESSIONHANDLE WinHandle
SET DIRECTORY	SET DIRECTORY Type <b>CREATE Path</b>
SET EMULATION*	SET EMULATION Emulation <b>WINDOW WinHandle</b>
SET FKEYSSHOW	SET FKEYSSHOW Boolean

---

## Syntax Quick Reference, *continued*

---

Command/Function Name	Statement Syntax
SET FLOWCONTROL*	SET FLOWCONTROL Type
SET KEEPPRINTCHANNELOPEN	SET KEEPPRINTCHANNELOPEN KBoolean
SET LOCALECHO*	SET LOCALECHO Boolean
SET NETID*	SET NETID Text
SET OUTGOINGCR*	SET OUTGOINGCR Option
SET PARITY*	SET PARITY Type
SET PASSTHROUGH*	SET PASSTHROUGH State
SET PASSWORD*	SET PASSWORD Text
SET PHONENUMBER*	SET PHONENUMBER PhoneNumber
SET RESULT	SET RESULT String
SET RETRY*	SET RETRY Boolean
SET RETRYDELAY*	SET RETRYDELAY Delay
SET SENDDELAY	SET SENDDELAY Delay
SET SIGNAL*	SET SIGNAL Boolean
SET SOUND	SET SOUND Boolean
SET STOPBITS*	SET STOPBITS Numeric
SET TERMCLOSE	SET TERMCLOSE Boolean <b>WINDOW WinHandle</b>
SET TERMFONT	SET TERMFONT FontName SizeX SizeY <b>WINDOW WinHandle</b>
SET USERID*	SET USERID Text
SET WILDCARD*	SET WILDCARD Str1 Str2
SET WINDOWTITLE	SET WINDOWTITLE String <b>WINDOW WinHandle</b>
SET WORDWRAP*	SET WORDWRAP Column
SET XCLOCK	SET XCLOCK Time <b>WINDOW WinHandle</b>
SET XSYSTEM	SET XSYSTEM Time <b>WINDOW WinHandle</b>
SETTINGS ( )	SETTINGS (Keyword, <b>WinHandle</b> )
SETTINGS	SETTINGS Tab <b>WINDOW WinHandle</b>
SHOW	SHOW
SPAWN	SPAWN <b>SAFE</b> Target <b>WINDOW WinHandle</b>
STR ( )	STR (Numeric, <b>Precision</b> )
SUBSTR ( )	SUBSTR (String, Start, <b>Length</b> )

---

## Syntax Quick Reference, *continued*

---

Command/Function Name	Statement Syntax
SWITCH	SWITCH SwitchVariable CASE Var1: <b>Command</b> <b>LEAVE</b> CASE Var2: <b>Command</b> <b>LEAVE</b> <b>DEFAULT:</b> <b>Command</b> <b>LEAVE</b> SWITCH END
SYSMETRICS ( )	SYSMETRICS ( )
SYSTEM ( )*	SYSTEM (SysNum P1, P2)
SYSTEM*	SYSTEM SysNum P1, P2, ...
-T-	
TABLE CLEAR	TABLE CLEAR Table
TABLE CLOSE	TABLE CLOSE Table
TABLE COPY	TABLE COPY SourceTable TO DestTable <b>Contents String</b>
TABLE DEFINE	TABLE DEFINE Table FIELDS f1...fi <b>FILE TEXT</b> FileName
TABLE LOAD	TABLE LOAD Table FROM Source AS Format
TABLE SAVE	TABLE LOAD Table FROM Source AS Format
TABLE SORT	TABLE SORT Table Fld1 Dir1 Fld2 Dir2 Fld3 Dir3
TASKERROR	TASKERROR Level Code
TIME ( )	TIME (Seconds)
TIMER ( )	TIMER (Index)
TIMER RESET	TIMER RESET Timer
TITLE	TITLE String
TOOLBARHIDE	TOOLBARHIDE BarName
TOOLBARSHOW	TOOLBARSHOW BarName

## Syntax Quick Reference, *continued*

Command/Function Name	Statement Syntax
TRANSFERS	TRANSFERS CommandProcessor BLOCKSIZE Status Length ISSUECLEAR Status PACKETSIZE Number HOSTPROGRAM FileName RECORDLENGTH Status Length RECORDFORMAT Type SPACE Status InitialSpace AddedSpace UNITS Type CPUUSAGE Number TIMEOUT Number HOSTCODEPAGE Country PCCODEPAGE Country WINDOW WinHandle
TRIM ( )	TRIM (String, Pre, Post)
TYPEDLIBRARYCALL ( )	TYPEDLIBRARYCALL (LibName, ProcName, TypeString, Param1, Param2, ..., Param <i>n</i> )
<b>–U–</b>	
UPPER ( )	UPPER (String)
USERID ( )*	USERID ( )
<b>–V–</b>	
VERSION ( )	VERSION ( )
VISIBLE ( )	VISIBLE (WinHandle)
<b>–W–</b>	
WAIT CHAR*	WAIT CHAR Character WINDOW WinHandle
WAIT CLOSE*	WAIT CLOSE WINDOW WinHandle
WAIT DELAY	WAIT DELAY Time
WAIT ECHO	WAIT ECHO WINDOW WinHandle
WAIT EDIT	WAIT EDIT (x, y, w, h) FileName
WAIT PROMPT*	WAIT PROMPT NumChar WINDOW WinHandle
WAIT QUIET	WAIT QUIET Time WINDOW WinHandle
WAIT RESUME	WAIT RESUME
WAIT SCREEN	WAIT SCREEN WINDOW WinHandle
WAIT STRING*	WAIT STRING String QUIET Time Window WinHandle
WAIT UNTIL	WAIT UNTIL Time
WHEN CANCEL	WHEN CANCEL CommandType
WHEN COLLECT	WHEN COLLECT Index OnStr OffStr FromStr ToStr WaitExclude Quiet limit SAVECR NOTTERMINAL WINDOW WinHandle
WHEN DISCONNECT	WHEN DISCONNECT WINDOW WinHandle Command
WHEN ECHO	WHEN ECHO StringVar WINDOW WinHandle Command

---

## Syntax Quick Reference, *continued*

---

Command/Function Name	Statement Syntax
WHEN ERROR	WHEN ERROR LevelNum Command
WHEN INPUT	WHEN INPUT StringVar <b>WINDOW</b> WinHandle Command
WHEN QUIET	WHEN QUIET Time <b>WINDOW</b> WinHandle Command
WHEN SCREEN	WHEN SCREEN Index (Row, Col, Wid, Lns) <b>WINDOW</b> WinHandle Command
WHEN STRING*	WHEN STRING Index String <b>WINDOW</b> WinHandle Command
WHEN TIMER	WHEN TIMER Time Command
WHEN WINDOW*	WHEN WINDOW WinHandleVar Message WParam LParam
WHILE	WHILE Boolean Command
WINDOW ( )	WINDOW (WinHandle)
WINDOWHND ( )	WINDOWHND (WinName)
WINDOWNAME ( )	WINDOWNAME (WinHandle)
WINDOW ACTIVATE	WINDOW ACTIVATE WinHandle <b>Boolean</b>
WINDOW ARRANGE	WINDOW ARRANGE
WINDOW CLOSE	WINDOW CLOSE WinHandle
WINDOW DEFAULT	WINDOW DEFAULT WinHandle Message WParam LParam
WINDOW HIDE	WINDOW HIDE <b>WinHandle</b>
WINDOW MAXIMIZE	WINDOW MAXIMIZE WinHandle
WINDOW MESSAGE	WINDOW MESSAGE WinHandle Message WParam LParam
WINDOW MINIMIZE	WINDOW MINIMIZE <b>WinHandle</b>
WINDOW MOVE	WINDOW MOVE <b>WinHandle (x, y, w, h)</b>
WINDOW OPEN	WINDOW OPEN Type FileName (x, y, w, h) <b>Boolean WinHandle</b>
WINDOW RESTORE	WINDOW RESTORE <b>WinHandle</b>
WINDOW STACK	WINDOW STACK
WINDOW UNHIDE	WINDOW UNHIDE <b>WinHandle</b>
WNDCLASS ( )	WNDCLASS (WinHandle)
WNDFILE ( )	WNDFILE (WinHandle)
WNDTITLE ( )	WNDTITLE (WinHandle)



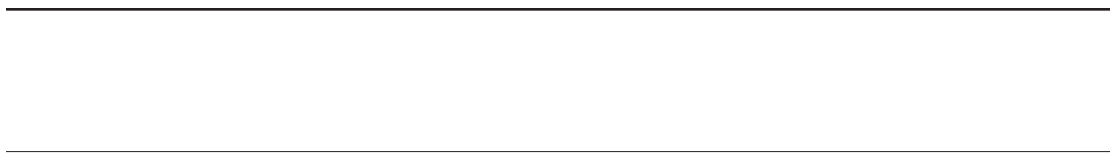
---

## Syntax Quick Reference, *continued*

---

Command/Function Name	Statement Syntax
-X-	
XFERCONFIG	XFERCONFIG String WINDOW WinHandle
-Z-	
ZOOMED ( )	ZOOMED (WinHandle)

---



---

# Index

---

## Index

### Symbols

- 3270 emulations
  - field attributes, retrieving 129, 138, 140
  - field identifier, retrieving 142
  - field length, retrieving 141
  - field starting position, retrieving 143
  - XCLOCK messages 514
  - XSYSTEM messages 515
- \$ (dollar sign)
  - named string variables 27
  - statement syntax 51
- + (addition operator or plus sign)
  - precedence 58
  - statement syntax 54
  - use of 38
- = (assignment operator)
  - statement syntax 55
  - use of 47
- \* (asterisk)
  - label statement syntax 49
  - use of 25
- @ (at sign)
  - use of 50
- \ (backslash)
  - embedding control characters in a string 27
  - statement syntax 50
  - use of 23
- .bmp files. *See* bitmap graphics.
- ^ (caret)
  - embedding control characters in a string 27
  - statement syntax 50
- : (colon)
  - use of 50
- , (comma)
  - statement syntax 51
  - use of 23
- | (concatenation operator)
  - creating string expressions with 35
  - statement syntax 58
  - use of 23
- .dcp (source script file), defined 17
- .dct (task script file), defined 17
- / (division operator)
  - precedence 58
  - statement syntax 54
  - use of 38
- ! (exclamation point)
  - statement syntax 51
  - use of 37
- @F (Function key variable)
  - statement syntax 50
  - use of 32
- > (greater than)
  - in relational expressions 40
  - precedence 58
  - statement syntax 55
- >= (greater than or equal to)
  - in relational expressions 40
  - precedence 58
  - statement syntax 56
- < (less than)
  - in relational expressions 40
  - precedence 58
  - statement syntax 55
- <= (less than or equal to)
  - in relational expressions 58
  - precedence 58
  - statement syntax 56
- % (modulus operator)
  - precedence 58
  - statement syntax 55
  - use of 38
- \* (multiplication operator)
  - precedence 58
  - statement syntax 54
  - use of 38
- != (not equal to)
  - in relational expressions 40
  - precedence 58
  - statement syntax 56
- ≠ (not equal to)
  - in relational expressions 40
  - precedence 58
  - statement syntax 56
- # (number sign)
  - statement syntax 51
  - use of 40, 348, 355
- = or == (equal to)
  - in relational expressions 40
  - precedence 58
  - statement syntax 55
- ( ) (parentheses)
  - precedence 58
  - statement syntax 51
  - use of 51
- % (percent sign)
  - statement syntax 52
  - use of 37
- . (period)
  - use of and statement syntax 52
- ? (question mark)
  - use of and statement syntax 52
- " (quotation mark)
  - embedding in a string 27
  - null string, creating 27
  - statement syntax 52
- @R (Record Buffer variable)
  - statement syntax 50, 66, 68
  - use of 28
- ;(semicolon)
  - use of 24, 53
- ' (single quote) 52
- @S (Settings variable)
  - statement syntax 50
  - use of 34

- (subtraction operator)  
 precedence 58  
 statement syntax 54  
 use of 38  
 @T (Function Key Title variable)  
 statement syntax 50  
 use of 32  
 - (unary minus)  
 precedence 58  
 use of 37  
 .wmf files. *See* Windows metafile graphics.

## A

ACCESS CANCEL (DDE command). *See* (DDE) ACCESS CANCEL command.  
 ACCESS (DDE command). *See* (DDE) ACCESS command.  
 ACTIVE function 96  
 active session, defined 74  
 active window, defined 74  
 addition operator (+)  
 precedence 58  
 statement syntax 54  
 use of 38  
 ADDS emulation, configuring.. *See* emulation configuration.  
 advise message  
 determining reception of 112  
 servicing requests via script 270  
 AND (Boolean operator)  
 precedence 58  
 use of 57  
 AND function. *See* BAND function.  
 ANSI emulation, configuring. *See* emulation configuration.  
 API (Application Programming Interface) calls 203  
 APPCONFIG command 240–241  
 appending selected text to a file 453  
 application configuration 85  
 applications, launching via script 393  
 application title, setting 465  
 arguments  
 Command 22  
 definition 20  
 field 28, 31  
 file names 43  
 Function 51  
 Key 32  
 Numeric 43  
 Table 28, 31, 43, 64, 67  
 types 43  
 valid content type 20  
 WinHandle 74  
 ARGUMENTS command 242  
 creating variables 47, 62  
 example of 61, 63  
 argument types 43  
 ASCII characters  
 converted to 346  
 embedding 27, 50

in CHR function 105  
 assigning values to variables 47, 59  
 assignment commands, grouped 227  
 assignment operator (=)  
 defined 21  
 precedence 58  
 use of 47  
 asterisk (\*)  
 as modulus operator 38  
 label statement syntax 49  
 use of 25  
 AT&T emulations, configuring. *See* emulation configuration.  
 attributes  
 of fields, retrieving 129, 138, 140  
 of files, retrieving 97  
 ATTRIBUTES function 97  
 autosizing font 468  
 autostart script, defined and use of 17

## B

backslash (\)  
 embedding control characters 27  
 statement syntax 50  
 use of 23  
 backspace key, setting 469, 470  
 BAND function 98  
 baud rate, setting 471  
 BEEP command 243  
 BEGIN command 244  
 example of 22, 45, 106, 501  
 bell (host), enabled state 506  
 bell (PC), ringing via script 243  
 bitmap graphics  
 adding to scripted dialog boxes 311  
 copyig to a printer 325  
 copying to the clipboard 325  
 bitwise  
 AND operation 98  
 complement operation 99  
 OR operation 101, 104  
 BNOT function 99  
 Boolean constants, statement syntax and use of 40  
 Boolean expressions  
 definition and use of 41  
 parameters, passing 61  
 Boolean functions  
 definition and use of 40  
 grouped 92  
 Boolean modifiers 57  
 Boolean operators  
 AND 57  
 NOT 57  
 OR 57  
 use of 41  
 Booleans  
 complex 42  
 Boolean value 100, 107, 112  
 Boolean variables, named  
 definition and use of 40, 47, 48  
 parameters, passing 62  
 BOOL function 100

---

BOR function 101  
 branching commands, grouped 227  
 BREAK command 245  
   use of 33  
 break, sending to a host 463  
 break signal, sending via script 245  
 BUFFER function 102  
 buffer, session. *See* history buffer.  
 BUTTON (Dialog command). *See* (DIALOG) BUTTON command.  
 buttons (in scripted dialog boxes)  
   creating 291  
   positioning and sizing 300  
   updating attributes 296,315  
 BXOR function 104

**C**

CANCEL command 246  
   example of 56  
 capturing data, functions for 92  
 capturing incoming data, command 248  
 caret (^)  
   embedding control characters, used to 27  
   statement syntax 50  
 carriage return, outgoing 496  
 carrier detection 476  
 cascading windows 586  
 channel numbers (DDE) 80  
 characters  
   copying to  
     a file 325  
     a printer 325  
     the clipboard 324  
   displaying in a session window 319  
   echo in session, detecting 548  
   print attributes 431  
   removing from a string 202  
   replacing in a string 136  
   waiting for specified 544,548  
 CHECKBOX (Dialog command). *See* (DIALOG) CHECKBOX command.  
 check boxes (in scripted dialog boxes)  
   creating 294  
   determining status of 116  
   positioning and sizing 300  
   updating attributes 296,315  
 child routine, defined 59  
 child windows 110  
 CHR function 105–106  
   example of 105  
 CLEAR command 247  
 clipboard  
   copying characters to 324  
   copying graphics to 325  
   cutting to 328  
   pasting text from 331  
 COLLECT command 248–250  
   example of 47  
   use of 47  
 colon (:), definition and use of 50  
 column width, setting 477  
 combo boxes. *See* list boxes.  
 comma (,)  
   statement syntax 51  
   use of 23  
 command block 51  
   defined and use of 22  
 command block commands, grouped 228  
 command blocks  
   beginning 244  
   branching to multiple 519  
   ending 343  
   for scripted menus 405  
 commands  
   assignment operations, grouped 227  
   branching operations, grouped 227  
   command block operations, grouped 228  
   conditional operations, grouped 227  
   configuration operations, grouped 228  
   continuing on another line 23  
   conversion operations, grouped 229  
   data and file transfer operations, grouped 229  
   DDE operations, grouped 231  
   defined 20  
   dialog operations, grouped 230  
   DLL operations, grouped 231  
   edit operations, grouped 231  
   file operations, grouped 232  
   listed with brief explanations 217,604  
   math operations, grouped 233  
   menu operations, grouped 233  
   multiple on the same line 23  
   network operations, grouped 233  
   new to DCS, listed 600  
   print operations, grouped 234  
   removed from DCS, listed 601  
   script control operations, grouped 234  
   session window operations, grouped 235  
   string operations, grouped 235  
   subroutine operations, grouped 227  
   system operations, grouped 236  
   table operations, grouped 236  
   telecommunications operations, grouped 237  
   toolbar operations, grouped 233  
 command syntax 20  
 comments, defined and syntax 24  
 COMPILE command 251  
 compiling scripts 17  
   caution note 19  
   errors 19  
   via script 251  
 complex  
   Booleans 42  
   numeric variable, defined and use of 38  
   string  
     creating 27  
     defined and use of 36  
 compressing files 346  
 CONCAT command 252  
 concatenation  
   of strings via script 36,252  
 concatenation operator (!)  
   statement syntax 58  
   use of 23  
 conditional branching, event-based 76

conditional commands, grouped 227  
 configuration commands, grouped 228  
 configuration functions, grouped 92  
 configuring
 

- application options via script 240
- applications 85
- connectors 84
- connectors via script 253, 478
- displays 84
- displays via script 321
- emulations 84
- emulations via script 334, 488
- file transfers 84
- file transfers via script 472, 474, 539, 588
- menus 85
- sessions 84
- sessions via script 516
- toolbar 85

 CONNCONFIG command 253–255  
 CONNECT command 256–257
 

- example of 56
- use of 74

 CONNECT function 107  
 connecting a session via script 256  
 connection status, determining 107  
 connection termination, detecting 561  
 CONNECTMESSAGE function 108  
 ConnectMessage system variable
 

- retrieving current value 108
- setting 480

 connector configuration 84, 478
 

- command and keywords 253
- retrieving current settings 146

 CONNECTRESULT function 109  
 ConnectResult system variable
 

- retrieving current value 109
- setting 481

 constant
 

- Boolean 40
- numeric 37

 CONTINUE command 258  
 continuing, line 23  
 control characters
 

- defined 43
- embedding in a string 27

 conversion commands, grouped 229  
 conversion functions, grouped 92  
 conversion of scripts 84  
 converting numeric variables 37  
 copying
 

- characters to clipboard 325
- files 347

 CREATE DIRECTORY command 259  
 critical errors 82  
 cursor
 

- appearance, setting 482
- automatic scrolling to, setting 467
- positioning at beginning of line 330

 CURSOR function 110  
 cursor position 110  
 cutting to the clipboard 328

data
 

- receiving from DDE server 268
- retrieving from screen display 102, 178
- searching and capturing functions, grouped 92
- sending to DDE server 264

 data bits, setting 483  
 data items (DDE) 80  
 data manipulation through tables 64  
 data types 28  
 DATE function 111  
 date, retrieving 111  
 DCS
 

- terminating via script 434
- version number, retrieving 207

 DDE. *See* DDE (Dynamic Data Exchange).  
 (DDE) ACCESS CANCEL command 262  
 (DDE) ACCESS command 260–261
 

- example of 81
- use of 80

 (DDE) ADVISE function 112  
 DDE (Dynamic Data Exchange)
 

- canceling wait state 557
- commands, grouped 231
- commands, sending to server 263
- concepts and use of 78
- conversation initiation, detecting 275
- conversation termination, detecting 280
- data
  - receiving from a server 265
  - sending to server 264
- DCS server name 79
- events, detecting 275, 276, 278, 280
- functions, grouped 93
- initiating via script 260
- maximum number of channels 80
- overview of 78
- servicing requests 271, 273, 276, 278
- statement syntax for messages 81
- structured table
  - sending to server 266, 269
  - storing received data 268
- terminating via script 262
- wait signal 270

 (DDE) INSTRUCT command 263
 

- example of 81

 (DDE) POKE command 264
 

- use of 80

 (DDE) REQUEST command 265
 

- use of 47, 80

 (DDE) TABLE REPLY command 266–267  
 (DDE) TABLE REQUEST command 268
 

- use of 80

 (DDE) TABLE SEND command 269
 

- use of 80

 (DDE) WAIT SIGNAL command 270  
 (DDE) WHEN ADVISE command 271–272  
 (DDE) WHEN EXECUTE command 273–274  
 (DDE) WHEN INITIATE command 275
 

- use of 80

 (DDE) WHEN POKE command 276–277  
 (DDE) WHEN REQUEST command 278–279  
 (DDE) WHEN TERMINATE command 280  
 DEBUG command 281–282  
 debug window

## D

---

- hiding 419
- showing 517
- decimal places, setting 485
- decompressing files 351
- DECREMENT command 283
- DECRYPT function 113–114
- decrypting files 352
- default directories, listed 123
- DEFAULTSESSIONHANDLE function 115
  - and multiple concurrent scripts 74
  - use of 75
- default session window handle, setting 486
- delaying script execution 546
  - during memo editing 549
  - waiting for a string 555
  - waiting for period of inactivity 551
  - waiting for RESUME command 553
  - waiting for screen refresh 554
  - waiting for specified file to close 545
  - waiting for specified number of characters 550
  - waiting for specified time of day 556
- destination argument. *See* file name arguments.
- detecting events during script execution 76
- dial
  - retries, setting 502
  - retry delay, setting 503
  - signal, setting 505
- DIAL command 284–285
  - example of 500, 502
  - use of 35
- dialog boxes, scripted
  - as child routines 59
  - bitmap graphics
    - adding 311
    - updating 315
  - buttons
    - creating 291
    - positioning and sizing 300
    - updating attributes 296, 315
  - check boxes
    - creating 294
    - positioning and sizing 300
    - updating attributes 296, 315
  - controls
    - creating 307
    - maximum number of 288
    - updating 296–299, 315
  - creating (command structure) 286
  - destroying 293
  - displaying
    - pictures 311
    - text 309
  - edit text boxes
    - creating 301
    - positioning and sizing 300
    - updating attributes 296, 315
  - group boxes
    - creating 303
    - positioning and sizing 300
    - updating attributes 296, 315
  - icon buttons
    - creating 305
    - positioning and sizing 300
    - updating attributes 296, 315
  - icons
    - creating 304
    - positioning and sizing 300
    - updating attributes 296, 315
  - line spaces, adding 310
  - list boxes
    - creating 307
    - positioning and sizing 300
    - updating attributes 296, 315
  - messages
    - creating 309
    - positioning and sizing 300
    - updating attributes 296, 315
  - modal vs. non-modal 287
  - radio buttons
    - creating 312
    - positioning and sizing 300
    - updating attributes 296, 315
  - radio groups
    - creating 313
    - positioning and sizing 300
    - updating attributes 296, 315
  - title bar and control menu 286
  - updating controls 296
  - widebuttons, updating attributes 296, 315
  - Windows metafile graphics
    - adding 311
    - updating 315
- (DIALOG) BUTTON command 291–292
- DIALOG CANCEL command 293
  - example of 35
- (DIALOG) CHECKBOX command 294–295
- (DIALOG) CHECKBOX function 116
- DIALOG command 286–290
- dialog commands
  - grouped 230
  - used in command blocks 22
- DIALOG CONTROL command 296–299
- (DIALOG) DIMENSION command 300
- (DIALOG) EDITTEXT command 301–302
  - example of 35
- (DIALOG) EDITTEXT function 117
- dialog functions, grouped 93
- (DIALOG) GROUPBOX command 303
- DIALOGHANDLE 122
- (DIALOG) ICONBUTTON command 305–306
- (DIALOG) ICON command 304
- (DIALOG) LISTBOX command 307–308
- (DIALOG) LISTBOX function 118
- (DIALOG) MESSAGEBOX function 119–120
- (DIALOG) MESSAGE command 309
- (DIALOG) NEWLINE command 310
- (DIALOG) PICTURE command 311
- (DIALOG) RADIOBUTTON command 312
- (DIALOG) RADIOGROUP command 313–314
- (DIALOG) RADIOGROUP function 121
- dialogs, Script Compiler 19
- DIALOG UPDATE command 315–317
- Digital VT emulations, configuring. *See* emulation configuration.
- DIMENSION (Dialog command). *See* (DIALOG) DIMENSION command.

- 
- dimensions, window, retrieving 167
- directories
- creating via script 259
  - default paths, listed 123
  - default paths, setting 487
  - deleting 441
- DIRECTORY function 123–124
- example of 97
- Direct Serial connector. *See also* connector configuration.
- baud rate, setting 471
  - data bits, setting 483
  - dialing via script 284
  - flowcontrol, setting 491
  - parity, setting 497
  - stopbits, setting 507
- direct variables, defined and use of 47
- DISCONNECT command 318
- disconnecting a session 318
- disk space, determining free remaining 194
- DISKSPACE function 125
- DISPLAY command 319–320
- example of 22, 31, 41, 45, 106, 109
  - use of 47
- DISPLAYCONFIG command 321–322
- display configuration 84
- autosize font, setting 468
  - command and keywords 321
  - cursor appearance 482
  - font size, setting 509
  - font type, setting 509
  - local echo, setting 494
  - retrieving current settings 147
- displaying characters in a session window 319
- division operator (/)
- precedence 58
  - statement syntax 54
  - use of 38
- DLL (Dynamic Link Library)
- API calls, retrieving values 203
  - calling 397
  - commands, grouped 231
  - loading 400
  - unloading 401
- dollar sign (\$)
- named string variables 27
  - statement syntax 51
- DROPDTR command 323
- DTR line 323
- dynamic data exchange. *See* DDE (Dynamic Data Exchange).
- Dynamic Data Exchange (DDE). *See* DDE (Dynamic Data Exchange).
- Dynamic Link Library (DLL). *See* DLL (Dynamic Link Library).
- E**
- edit commands, grouped 231
- EDIT COPY command 324
- EDIT COPYSPECIAL command 325–327
- EDIT CUT command 328
- EDIT FIND command 329
- EDIT GOTO command 330
- editing scripts 18
- EDIT PASTE command 331
- EDIT REPLACE command 332–333
- edit text boxes (in scripted dialog boxes)
- creating 301
  - positioning and sizing 300
  - retrieving contents of 117
  - updating attributes 296, 315
- EDITTEXT (Dialog command). *See* (DIALOG) EDITTEXT command.
- ELSE clause
- example of 23
  - used in command blocks 22
- embedding characters in a string 27
- emulation configuration 84, 488
- command and keywords 334
  - retrieving current settings 148
- EMULCONFIG command 334–342
- encrypted strings, decrypting 113
- ENCRYPT function 126
- encrypting 126
- encrypting files 354
- encryption key. *See* ENCRYPT function.
- END command 343
- example of 22, 501
  - used in command blocks 22
- end-of-file. *See* EOF function.
- EOF function 127
- example of 45, 46, 57
- equal to operator (= or ==)
- in relational expressions 40
  - precedence 58
  - statement syntax 55
- error detection 563
- ERROR function 128
- example of 46, 49
  - returns 97, 107, 110
  - use of 82
- error numbers. *See also* TASKERROR command.;  
*See* task errors.
- errors 82
- errors, determining occurrence of 128
- escape sequences, representing in a string 27
- evaluation order 51, 55
- event handling, defined and use of 76
- events 76, 77
- exclamation point (!)
- statement syntax 51
  - use of 37
- EXECUTE command 344
- use of 33, 44
- executing scripts 18
- EXFLDATTR function 129–133
- existence of a file, determining 134
- EXISTS function 134
- exponential function 168. *See* POWER function.
- expressions
- Boolean 41
  - defined 21



---

described 21  
numeric, creating 38  
relational 40  
string 35

## F

far target arguments, definition and use of. *See* labels.  
far targets

definition and use of 45  
executing 422  
returning from 445

fatal errors 82

field functions, grouped 93

fields

attributes, retrieving 129, 138, 140  
defining  
for structured tables 64  
for text tables 67  
identifier, retrieving 142  
length, retrieving 141  
starting position, retrieving 143

file attributes 97

FILE CLOSE command 345

FILE COMPRESS command 346

FILE COPY command 347

FILE CREATENAME command 348–350  
use of 47

FILE DECOMPRESS command 351

FILE DECRYPT command 352

FILE DELETE command 353

FILE ENCRYPT command 354

file name arguments, definition and use of 43

FILE OPENNAME command 355–356  
use of 47

FILE PAUSE command 357

FILE RECEIVE BINARY command 358–360

FILE RENAME command 361

FILE RESUME command 362

files

appending selected text to 453  
attributes as numeric values, listed 97  
attributes, setting 466  
commands, grouped 232  
compressing 346  
copying 347  
creating 348  
decrypting 352  
deleting 353  
encrypting 354  
existence, determining 134  
functions, grouped 93  
logging incoming data to 404  
name, retrieving from associated window 213  
opening 355, 583  
path names, representing 43  
renaming 361  
retrieving default path by type 123  
saving 446  
session, loading 403  
size, determining 135  
uncompressing 351

FILE SEND BINARY command 363–365

FILESIZE function 135

file transfer commands, grouped 229, 232

file transfer configuration 84, 474, 539, 588  
retrieving current settings 151  
word wrap, setting 513

file transfers

configuring 84, 472, 474, 539, 588  
receiving binary files 358  
resuming suspended 362  
sending binary files 363  
suspending 357  
terminating 345  
waiting for specified file to close 545

FILTER function 136–137

find or finding. *See* searching.

FKEYS command 366

FLDATTREXPOS function 140

FLDATTR function 138–139

FLDLEN function 141

FLDNUM function 142

FLDPOS function 143

FLDTEXT function 144

flow control, setting 491

font configuration. *See* display configuration.

free disk space, determining 125

Function Key Title variable (@T)

maximum characters 32  
statement syntax 50

Function Key variable (@F)

example of 50  
maximum characters 32  
statement syntax 50

functions

Boolean 40  
Boolean operations, grouped 92  
configuration operations, grouped 92  
conversion operations, grouped 92  
data searching and capturing operation, grouped 92  
DDE operations, grouped 93  
defined 20  
dialog operations, grouped 93  
field operations, grouped 93  
file operations, grouped 93  
listed with brief explanations 88, 604  
math operation, grouped 94  
menu operations, grouped 94  
new to DCS, listed 600  
numeric 37  
removed from DCS, listed 602  
string operations, grouped 94  
system operations, grouped 94  
table operations, grouped 95  
telecommunication operations, grouped 95  
window operations, grouped 95

function syntax 20

## G

GENERALCONFIG command 367–368

GETAPPCONFIG function 145

GETCONNCONFIG function 146  
 GETDISPLAYCONFIG function 147  
 GETEMULCONFIG function 148  
 GETGENERALCONFIG function 149  
 GETPROFILEDATA function 150  
 GETXFERCONFIG function 151–152  
 GOTO command 369  
   example of 23, 46, 107  
   use of 44  
 greater than operator (>)  
   in relational expressions 40  
   precedence 58  
   statement syntax 55  
 greater than or equal to operator (>=)  
   in relational expressions 40  
   precedence 58  
   statement syntax 56  
 GROUPBOX (Dialog command). *See* (DIALOG) GROUPBOX command.  
 group boxes (in scripted dialog boxes)  
   creating 303  
   positioning and sizing 300  
   updating attributes 296, 315  
 grouped. *See* radio groups (in scripted dialog boxes).

## H

HANGUP command 370  
 hidden windows, effect on commands 74  
 history buffer  
   saving selected text 456  
   scrolling  
     down 448  
     left 449  
     right 450  
     up 451  
   selected text, sending 457  
   selecting text in 454  
   size, setting 475  
 host cursor position, retrieving 110  
 HWNLIST function 153–154  
   use of 74

## I

ICONBUTTON (Dialog command). *See* (DIALOG) ICONBUTTON command.  
 icon buttons (in scripted dialog boxes)  
   creating 305  
   positioning and sizing 300  
   updating attributes 296, 315  
 ICON (Dialog command). *See* (DIALOG) ICON command.  
 ICONIC function 155  
 icons (in scripted dialog boxes)  
   creating 304  
   positioning and sizing 300  
   updating attributes 296, 315  
 IF command 371  
   example of 22, 23, 41, 83, 107, 112, 116  
 IF - ELSE command block, across multiple lines 23  
 Include Line Numbers 19

INCREMENT command 372  
   example of 22, 31, 48, 70, 106  
 indirect variables, definition and use of 48  
 initialization files  
   creating or changing entries 170  
   retrieving entry text 150  
 integer numeric variable  
   converting to real variable 37  
   defined 37  
   example of 38  
 integers  
   converting  
     to real numerics 173  
 intergers  
   converting  
     from a real numeric 156  
     from a string 163  
     to Boolean values 100  
 INT function 156  
   use of 37  
 ITEM (Menu command). *See* (MENU) ITEM command.

## K

keep print channel open, setting 492  
 Kermit configuration. *See* file transfer configuration.  
 KERMIT COPY command 373  
 KERMIT DIRECTORY command 374  
 KERMIT ERASE command 375  
 KERMIT FINISH command 376  
 KERMIT FREESPACE command 377  
 KERMIT HELP command 378  
 KERMIT LOGOUT command 379  
 KERMIT MESSAGE command 380  
 KERMIT NEWDIRECTORY command 381  
 KERMIT RENAME command 382  
 KERMIT TYPE command 383  
 KERMIT WHO command 384  
 keyboard  
   input, detecting 564  
   remapping via script 385  
   resetting 391  
   state, resetting 389  
 KEYBOARD command 389  
 KEY command 385–388  
   example of 105  
   use of 44  
 key map files  
   loading 390  
   resetting 391  
   saving 392  
 KEYMAP LOAD command 390  
 KEYMAP RESET command 391  
 KEYMAP SAVE command 392  
 keywords  
   CHAR, INT and REAL in TABLE DEFINE command 29  
   FILE in TABLE DEFINE command 64  
   TEXT in TABLE DEFINE command 67

## L

---

labels  
 defined and use of 25  
 definition and use of. *See* far target arguments; near target arguments.  
 statement syntax 49  
 LAUNCH command 393–394  
 example of 43  
 LEAVE command 395  
 length  
 of fields 141  
 LENGTH function 157  
 example of 36  
 length of a string, determining 157  
 less than operator (<)  
 in relational expressions 40  
 precedence 58  
 statement syntax 55  
 less than or equal to operator (<=)  
 in relational expressions 40  
 precedence 58  
 statement syntax 56  
 LEVEL command 396  
 example of 33  
 use of 32  
 LIBRARY CALL command 397–399  
 LIBRARY LOAD command 400  
 LIBRARY UNLOAD command 401  
 line continuation 23, 50  
 line numbers, adding to scripts 19, 402  
 LINENUMBERS command 402  
 LISTBOX (Dialog command). *See* (DIALOG) LISTBOX command.  
 list boxes (in scripted dialog boxes)  
 creating 307  
 determining currently selected item 118  
 positioning and sizing 300  
 updating attributes 296, 315  
 LOAD command 403  
 example of 35, 500  
 local echo, setting 494  
 logical structures 22  
 LOGTOFILE command 404  
 loop structure, creating 573

**M**

math  
 commands, grouped 233  
 functions, grouped 94  
 operators, listed 54  
 maximized window state, determining 215  
 maximizing a window 447, 579  
 maximum characters  
 for capturing incoming data 248  
 for function key title variables 32  
 for Function Key variables 32  
 for named Boolean variables 40  
 for named numeric variables 37  
 in a named string variable 27  
 in a structured table field 64  
 in Record Buffer variables 27  
 maximum number of  
 controls in a dialog box 288  
 DDE channels 80  
 fields in a structured table 28  
 tables 28, 64  
 memory, free remaining, determining 194  
 MENU CANCEL command 406  
 example of 72  
 (MENU) CHECKED function 158  
 MENU command 405  
 example of 73  
 menu commands  
 grouped 233  
 working with 71  
 MENU DELETE ITEM command 407  
 MENU DELETE POPUP command 408  
 menu editor 71  
 (MENU) ENABLED function 159  
 menu functions, grouped 94  
 MENU INSERT ITEM command 409–410  
 MENU INSERT POPUP command 411  
 (MENU) ITEM command 412–413  
 example of 73  
 menu items  
 determining if checked 158  
 determining if enabled 159  
 (MENU) POPUP command 414–415  
 example of 73  
 (MENU) SEPARATOR command 416  
 example of 73  
 menus, scripted  
 configuration via script 85  
 creating 405  
 destroying 406  
 items, adding 412  
 items, deleting 407  
 popup menus, adding 411, 414  
 popup menus, deleting 408  
 popup menus, example of 73  
 separators, adding 416  
 updating 417  
 MENU UPDATE command 417–418  
 Meridian LAT32 connector, configuring. *See* connector configuration.  
 messages. *See also* messages (in scripted dialog boxes).  
 determining response with 112  
 displaying 119  
 messages (in scripted dialog boxes)  
 creating 309  
 positioning and sizing 300  
 updating attributes 296, 315  
 META keys 33  
 MIDI application 71  
 minimized window state, determining 155  
 minimizing a window 581  
 modal dialog boxes 287  
 modem  
 dialing via script 284  
 disconnecting 370  
 Modem connector, configuring. *See* connector configuration.  
 modular scripts  
 creating with labels 25

use of parent and child routines 61  
 modulus operator (%)  
   precedence 58  
   statement syntax 55  
   use of 38  
 multiple concurrent scripts  
   launching 518  
   programming considerations 75  
   running 84  
 multiplication operator (\*)  
   precedence 58  
   statement syntax 54  
   use of 38

## N

named Boolean variables 40  
 named numeric variables 37  
 named string variables  
   defined and use of 27  
   maximum characters 27  
 near targets  
   arguments, defined and use of 44  
   executing 422  
   returning from 445  
 NETID function 160  
 network commands, grouped 233  
 Network ID system variable  
   setting 495  
 NetworkID system variable  
   retrieving current value 160  
 new commands, listed 600  
 new functions, listed 600  
 NEWLINE (Dialog command). *See* (DIALOG) NEWLINE  
   command.  
 new scripts, creating 17  
 NEXT function 161  
 NOSHOW command 419  
 NOT (Boolean operator)  
   example of 41  
   precedence 58  
   use of 57  
 not equal to operator (<> or !=)  
   in relational expressions 40  
   precedence 58  
   statement syntax 56  
 null string, creating 27  
 number sign (#)  
   use of 40, 51, 348, 355  
 numeric constants, defined 37  
 numeric expressions  
   creating 38  
   parameters, passing 61  
 numeric functions, defined 37  
 numeric operand 37  
 numerics 21, 37, 38, 50  
   converting to strings 105, 192  
 numeric variables, complex  
   defined and use of 38  
 numeric variables, named

  defined and use of 37, 47  
   parameters, passing 61  
 NUM function 162  
   example of 30

## O

operands  
   Boolean 40  
   defined 21  
   multiple on the same line 23  
   numeric 37, 52  
   strings, defined 27  
   types 21  
 operations  
   AND 98  
   OR 101, 104  
   table 65  
 operator characters  
   defined and use of 54  
   precedence, listed 58  
 operators  
   assignment 47, 55  
   concatenation 36  
   math 54  
   precedence 58  
   relational 55, 56  
   using 38  
 optional syntax, notation convention 20  
 OR (Boolean operator)  
   example of 57  
   precedence 58  
   use of 57  
 ORD function 163  
 outgoing carriage return, setting 496

## P

parameter lists 51  
 parameter passing (DLL) 397  
 parameters  
   passing between child and parent routines 61  
   working with 61  
 parentheses ( ( ) ), use of and statement syntax 51  
 parent routine, defined 59  
 parity, setting 497  
 PARSE command 420–421  
   use of 47  
 passing variable parameters 61  
 passthrough printing, setting 498  
 PASSWORD function 164  
 Password system variable  
   retrieving current value 164  
   setting 499  
 pasting from the clipboard 331  
 path and file names, representing 43  
 percent sign (%)  
   as modulus operator 38  
   statement syntax 55  
   use of 37

PERFORM command 422  
 compared with SPAWN command 518  
 example of 45, 63, 112, 116, 495  
 use of 33, 44  
 period (.), use of and statement syntax 52  
 PHONENUMBER function 165  
 Phonenum system variable  
 retrieving current value 165  
 setting 500  
 PICTURE (Dialog command). *See* (DIALOG) PICTURE command.  
 piping symbol (|). *See* concatenation operator.  
 plus sign (+). *See* addition operator (+).  
 POKE (DDE command). *See* (DDE) POKE command.  
 POPUP (Menu command). *See* (MENU) POPUP command.  
 POS function 166  
 position  
 of a window, determining 167  
 of first character in a string, determining 166  
 of host cursor, determining 110  
 POSITION function 167  
 pound sign (#). *See* number sign (#).  
 POWER function 168  
 precedence of operators, listed 58  
 PRINT CANCEL command 423  
 PRINT CLOSE command 424  
 PRINT FILE command 425  
 example of 498  
 PRINT FONT command 426  
 printing  
 carriage return or line feed 427  
 character attributes, setting 431  
 closing print channel 424  
 commands, grouped 234  
 file to printer 425  
 form feed 428  
 keep print channel open, setting 492  
 parameters, retrieving 169  
 passthrough state, setting 498  
 print channel, opening 429  
 printer font, changing 426  
 screen data, routing to printer 433  
 selected window area 455  
 string to printer 430  
 tab widths, setting 432  
 terminating via script 423  
 PRINT NEWLINE command 427  
 PRINT NEWPAGE command 428  
 PRINT OPEN command 429  
 PRINT STRING command 430  
 PRINT STYLE command 431  
 PRINT TABS command 432  
 PRINT TERMINAL command 433  
 programming considerations  
 for multiple concurrent scripts 75  
 PRMTRICS function 169  
 PUTPROFILEDATA function 170–171

## Q

question mark (?), use of and statement syntax 52

QUIT command 434  
 quotation mark (" or ')  
 embedding in a string 27  
 null string, creating 27  
 statement syntax 52

## R

RADIOBUTTON (Dialog command). *See* (DIALOG) RADIOBUTTON command.  
 radio buttons (in scripted dialog boxes)  
 creating 312  
 grouped. *See* radio groups.  
 positioning and sizing 300  
 updating attributes 296, 315  
 RADIOGROUP (Dialog command). *See* (DIALOG) RADIOGROUP command.  
 radio groups (in scripted dialog boxes)  
 creating 313  
 determining currently selected button 121  
 positioning and sizing 300  
 updating attributes 296, 315  
 RANDOM function 172  
 random numbers, generating 172  
 REAL function 173  
 use of 37  
 real numerics  
 converting  
 from an integer 173  
 from a string 162  
 to integers 156  
 decimal places, setting 485  
 exponential function 168  
 rounding operation 175  
 real numeric variable  
 converting to integer variable 37  
 defined 37  
 example of 38  
 Record Buffer variable (@R)  
 use of 28  
 Record Buffer variables  
 definition and use of 27  
 examples of 30  
 for structured tables 28  
 for text tables 28  
 maximum character 27  
 statement syntax 28, 50  
 RECORD FORMAT command 435  
 use of 66  
 RECORD READ command 436–437  
 example of 24, 31, 45, 66, 118  
 use of 68  
 RECORD SCAN command 438  
 use of 66  
 RECORD WRITE command 439–440  
 example of 31, 67, 70, 113  
 relational expressions, definition and use of 40  
 remapping keys via script 385  
 removed commands, listed 601  
 removed functions, listed 602  
 REMOVE DIRECTORY command 441  
 REQUEST (DDE command). *See* (DDE) REQUEST

command.  
 RESETSERIAL command 442  
 RESTART command 443  
 RESULT function 174  
 Result system variable  
   setting 501  
 Result sytem variable  
   current value, retrieving 174  
 RESUME command 444  
   example of 22, 76  
 resuming script execution 444  
   due to keyboard input 564  
   due to received character 562  
   due to received string 569  
   due to session disconnect 561  
   due to specified elapsed time 571  
   due to specified period of inactivity 566  
   due to specified screen activity 567  
   due to specified Windows message 572  
 retry delay, setting 503  
 RETURN command 445  
   example of 45  
 ROUND function 175  
 ROUTE function 176–177  
   example of 52  
 rules, scoping 59, 60  
 run-time errors 82

## S

SAVE command 446  
   example of 34  
 scoping rules 59, 60  
 SCREEN command 447  
 SCREEN function 178–179  
 Script Compiler dialog 19  
 script control commands, grouped 234  
 scripts  
   autostart 17  
   compiling and executing 18  
   compiling via script 251  
   converting 84, 85  
   converting or updating from previous versions 84  
   creating 17  
   delaying execution 546  
   editing 17  
   launching from a script 518  
   modular 61  
   multiple concurrent, writing and running 84  
   multiple sessions 84, 85  
   restarting 443  
   resuming execution 444  
   saving 18  
   starting from within scripts 344  
   stopping 434  
   stopping via script 246  
   title, assigning 536  
 scroll buffer. *See* history buffer.  
 SCROLL DOWN command 448  
 SCROLL LEFT command 449  
 SCROLL RIGHT command 450  
 SCROLL UP command 451  
 SEARCH function 180–181  
   example of 51, 56  
 searching a window 329  
 searching data, functions for 92  
 searching for and replacing characters 332  
 SEARCHINRECT function 182  
 SECONDS function 183  
   example of 55  
 SELECTION APPEND command 453  
 SELECTION BUFFER command 454  
 SELECTION command 452  
 SELECTION PRINT command 455  
 SELECTION SAVE command 456  
 SELECTION SEND command 457  
 semicolon (;)  
   statement syntax 53  
   use of 24  
 SENDBREAK command 463  
 SEND command 458–462  
   example of 35, 56, 105  
 sending delay, setting 504  
 sending text to a host 458  
 SEPARATOR (Menu command). *See* (MENU) SEPARATOR command.  
 separators 52  
 serial port, resetting 442  
 session  
   break, sending to a host 463  
   buffer. *See* history buffer.  
   closing of, restricting 508  
   column width, setting 477  
   connecting via script 256  
   connection status, determining 107  
   default window handle, setting 486  
   disconnecting 318  
   DTR and serial port 323  
   history buffer. *See* history buffer.  
   loading a saved file 403  
   saving 446  
   selected text, appending to file 453  
   selected text, saving 456  
   selected text, sending 457  
   selecting text 452  
   sending text to a host 458  
   termination, detecting 561  
 session buffer. *See* history buffer.  
 session configuration 84, 516  
   command and keywords 367  
   current settings, file transfer, retrieving 184–187  
   current settings, general, retrieving 184–191  
 Session Properties dialog, displaying 516  
 sessions, configuring 84, 85  
 session toolbars  
   displaying 490  
   display, state 366  
   levels, assigning 396  
 session window commands, grouped 235  
 session windows 109, 115  
 SET APPTITLE command 465  
 SET ATTRIBUTES command 466  
 SET AUTOSCROLLTOCURSOR command 467  
 SET AUTOSIZE command 468

---

SET BACKSPACEDESTRUCTIVE command 469  
 SET BACKSPACEKEY command 470  
 SET BAUDRATE command 471  
 SET BINARYTRANSFERPARAMS command 472–473  
 SET BINARYTRANSFERS command 474  
 SET BUFFERLINES command 475  
 SET CARRIERDETECT command 476  
 SET COLUMNS command 477  
 SET command 464  
   example of 51  
   use of 21, 47, 54  
 SET CONNECTION command 478–479  
 SET CONNECTMESSAGE command 480  
 SET CONNECTRESULT command 481  
 SET CURSOR command 482  
 SET DATABITS command 483  
 SET DDETIMEOUT command 484  
 SET DECIMAL command 485  
 SET DEFAULTSESSIONHANDLE command 486  
   and multiple concurrent scripts 75  
   use of 74, 75  
 SET DIRECTORY command 487  
 SET EMULATION command 488–489  
 SET FKEYSSHOW command 490  
 SET FLOWCONTROL command 491  
 SET KEEPPRINTCHANNELOPEN command 492–493  
 SET LOCALECHO command 494  
 SET NETID command 495  
 SET OUTGOINGCR command 496  
 SET PARITY command 497  
 SET PASSTHROUGH command 498  
 SET PASSWORD command 499  
 SET PHONENUMBER command 500  
 SET RESULT command 501  
 SET RETRY command 502  
 SET RETRYDELAY command 503  
 SET SENDEDELAY command 504  
 SET SIGNAL command 505  
 SET SOUND command 506  
 SET STOPBITS command 507  
 SET TERMCLOSE command 508  
 SET TERMFONT command 509  
 SETTINGS command 516  
 SETTINGS function 184–191  
 settings variables (@S)  
   defined and use of 34  
   statement syntax 50  
 SET USERID command 510  
 SET WILDCARD command 511  
 SET WINDOWTITLE command 512  
 SET WORDWRAP command 513  
 SET XCLOCK command 514  
 SET XSYSTEM command 515  
 SHOW command 517  
   use of 19  
 size of a file, determining 135  
 size of scripted dialog box 300  
 sounds, host enabled state 506  
 source argument. *See* file name arguments.  
 source script file, defined 17  
 SPAWN command 518  
   and multiple concurrent scripts 75  
 startup scripts 17  
 stopbits, setting 507  
 stopping script execution 246  
 STR function 192  
   example of 55  
 string constants, defined and use of 27  
 string expressions  
   creating 27  
   definition and use of 35  
   parameters, passing 62  
 string functions, grouped 94  
 strings 21, 27–36  
   as expressions, defined and use of 35  
   capturing incoming data into 248  
   commands, grouped 235  
   complex 36  
   complex, defined and use of 36  
   concatenation via script 252  
   constants 27  
   converting from  
     a numeric 105, 192  
   converting to  
     integers 163  
     real numerics 162  
     uppercase 205  
   creating complex 27  
   displaying in a session window 319  
   functions 27  
   functions, defined and use 27  
   in scripted dialog boxes. *See* messages (in scripted dialog boxes).  
   joining via script 252  
   length, determining 157  
   mapping to a key 385  
   parsing 420  
   position of first character, determining 166, 180, 182  
   printing 430  
   removing characters from 202  
   replacing characters 136  
   searching for 182, 329  
   searching for and replacing 332  
   sending to a host 458  
   wildcard characters, setting 511  
 string variable, named  
   defined and use of 47  
   definition and use of 27  
   parameters, passing 61  
 structured tables  
   clearing of data 523  
   data, importing 529  
   defined 64  
   field data types 28  
   manipulating records 66  
   manipulating whole table 65  
   maximum characters in a field 64  
   maximum number of fields 28  
   reading data from 66, 436  
   Record Buffer variables 27, 30  
   record format, creating 435  
   saving 67  
   saving to clipboard 530  
   saving to file 530  
   sending to DDE server 266, 269

sorting 531  
 statement syntax 64  
 writing data to 67, 438, 439  
 subroutine commands, grouped 227  
 subroutines 33, 61, 62, 63  
   executing 422  
   returning from 445  
 SUBSTR function 193  
 subtraction operator (-)  
   precedence 58  
   statement syntax 54  
   use of 38  
 SWITCH command 519–520  
 symbols 50  
   \$ (dollar sign) 51  
   " and ' (quotation marks) 52  
   @ (at sign) 50  
   \ (backslash) 50  
   ^ (caret) 50  
   characters, definition and use of 49  
   , (comma) 51  
   ! (exclamation point) 51  
   # (number sign) 51  
   () (parentheses) 51  
   % (percent sign) 52  
   . (period) 52  
   ? (question mark) 52  
   ; (semicolon) 53  
 syntax  
   quick reference 604  
   statement 20  
 SYSMETRICS function 194  
 system bell (PC), ringing via script 243  
 SYSTEM command 521–522  
 system commands, grouped 236  
 SYSTEM function 195–196  
 system functions, grouped 94  
 system parameters  
   retrieving 194, 195  
   setting 521  
 system variables 47  
   ConnectMessage 108  
   ConnectResult 109  
   NetworkID 160  
   Password 164  
   PhoneNumber 165  
   Result 174  
   UserID 206

## T

table arguments, definition and use of 43  
 TABLE CLEAR command 523  
 TABLE CLOSE command 524  
 table commands, grouped 236  
 TABLE COPY command 525–526  
 TABLE DEFINE command 527–528  
   example of 23, 30, 31, 57, 65, 113  
   for structured table 29, 65  
   use of 28, 64, 68  
   use of keywords 28, 64, 67  
 table field 52

table functions, grouped 95  
 TABLE LOAD command 529  
   example of 31, 57  
 TABLE REPLY (DDE command). *See* (DDE) TABLE REPLY command.  
 TABLE REQUEST (DDE command). *See* (DDE) TABLE REQUEST command.  
 tables. *See* structured tables; text tables.  
   clearing of data 523  
   closing 524  
   copying data 525  
   defined and use of 64  
   defining 527  
   end of, determining 127. *See also* EOF function.  
   manipulating structured tables 65  
   maximum number 28, 64  
   reading data from 436  
   record format, creating 435  
   structured 28, 64  
   saving 67  
   text 28, 67  
   use of (overview) 64  
   writing data to 438, 439  
 TABLE SAVE command 530  
   use of 67  
 TABLE SEND (DDE command). *See* (DDE) TABLE SEND command.  
 TABLE SORT command 531  
 targets  
   far 33, 45  
   near 44  
 TASKERROR command 532  
   example of 501  
 task errors  
   defined 82  
   listed with brief explanations 595  
 TASKFILE ii–xiv  
 task script file, defined 17  
 telecommunication  
   commands, grouped 237  
   functions, grouped 95  
 TeleVideo emulations, configuring.. *See* emulation configuration.  
 Telnet connector, configuring.. *See* connector configuration.  
 terminating file transfers 345  
 text (in scripted dialog boxes.. *See* messages (in scripted dialog boxes).  
 text tables  
   clearing of data 523  
   defining 67, 527  
   manipulating records 68  
   manipulating whole table 68  
   maximum record length 67  
   reading data from 68, 436  
   Record Buffer variables 28  
   statement syntax 67  
   writing data to 70, 438  
 tiling windows 575  
 time  
   determining elapsed 201  
   retrieving a value 197, 198, 199, 200  
 TIME function 200  
 TIMER function 201



---

TIMER RESET command 535  
TITLE command 536  
toolbar commands, grouped 233  
toolbar configuration via script 85  
TOOLBARHIDE command 537  
toolbars  
  hiding 537  
  showing 538  
TOOLBARSHOW command 538  
TRANSFERS command 539–543  
TRIM function 202  
troubleshooting 563  
  DEBUG command 281  
  debug window, hiding 419  
  debug window, showing 517  
TYPEDLIBRARYCALL function 203–204

## U

unadvise message, determining reception of 112  
unary minus operator (-)  
  precedence 58  
  use of 54  
uncompressing files 351  
updating scripts 84  
UPPER function 205  
USERID function 206  
UserID system variable  
  current value, retrieving 206  
  setting 510

## V

variables  
  assigning values via script 464  
  Boolean 40, 47, 51  
  branching, effect of 59  
  complex numeric 38  
  created in child routine 60  
  creating 47  
  creation of 59  
  DDE data, storing 265  
  declaring, command 242  
  decreasing numeric values 283  
  default 74  
  DEFAULTSESSIONHANDLE 74  
  direct 47  
  example of 21  
  Function Key (@F) 32  
  Function Key Title (@T) 32  
  increasing numeric values 372  
  indirect 27, 48  
  named Boolean 40  
  named numeric, defined and use of 37  
  named string 27, 51  
  numeric 36, 37, 47, 51, 52  
  passing between routines 61  
  record buffer 27, 30, 52  
  Result 174  
  scoping rules 59, 60  
  settings 34

  string 35  
  system 47, 108, 109, 160, 164, 165, 206  
VERSION function 207  
version of DCS, determining 207  
VISIBLE function 208  
VT Series emulations, configuring. *See* emulation configuration.

## W

WAIT CHAR command 544  
WAIT CLOSE command 545  
WAIT commands, using 76  
WAIT DELAY command 546–547  
WAIT ECHO command 548  
WAIT EDIT command 549  
WAIT PROMPT command 550  
WAIT QUIET command 551–552  
  example of 56  
WAIT RESUME command 553  
  example of 76  
  use of 79  
WAIT SCREEN command 554  
WAIT SIGNAL (DDE command). *See* (DDE) WAIT SIGNAL command.  
wait state  
  defined 76  
  terminating 76  
WAIT STRING command 555  
  example of 35  
WAIT UNTIL command 556  
warning errors  
  defined 82  
WHEN ADVISE (DDE command). *See* (DDE) WHEN ADVISE command.  
WHEN CANCEL command 557–558  
WHEN COLLECT command 559–560  
WHEN commands  
  use of 22, 76  
WHEN DISCONNECT command 561  
WHEN ECHO command 562  
  use of 47  
WHEN ERROR command 563  
  example of 501  
WHEN EXECUTE (DDE command). *See* (DDE) WHEN EXECUTE command.  
WHEN INITIATE (DDE command). *See* (DDE) WHEN INITIATE command.  
WHEN INPUT command 564–565  
  use of 47  
WHEN POKE (DDE command). *See* (DDE) WHEN POKE command.  
WHEN QUIET command 566  
  example of 22  
WHEN REQUEST (DDE command). *See* (DDE) WHEN REQUEST command.  
WHEN SCREEN command 567–568  
WHEN STRING command 569–570  
  example of 76  
WHEN TERMINATE (DDE command). *See* (DDE)

WHEN TERMINATE command.  
 WHEN TIMER command 571  
   example of 76  
 WHEN WINDOW command 572  
 WHILE command 573  
   across multiple lines 23  
   example of 24, 31, 45, 106  
   use of 22  
 wide buttons (in scripted dialog boxes)  
   updating attributes 296, 315  
 wildcard characters  
   use of 52  
 wildcard characters, setting 511  
 window  
   application title, setting 465  
   associated file name, retrieving 214  
   buffer. *See* history buffer.  
   cascading 586  
   clearing via script 247  
   closing 576  
   commands, grouped 239  
   default for Windows messages, setting 577  
   dimensions, retrieving 167  
   dimensions, setting 447  
   functions, grouped 95  
   hiding 578  
   making active 574  
   maximized state, determining 215  
   maximized state, setting 447  
   maximizing 579  
   minimized state, determining 155  
   minimizing 581  
   moving 582  
   name, retrieving 214, 215  
   name, setting 512  
   opening 583  
   pasting text from the clipboard 331  
   position, retrieving 167  
   restoring previous state 447  
   restoring to previous size 585  
   selected text, saving 456  
   selected text, sending 457  
   selecting text 452  
   showing (unhiding) 587  
   tiling 575  
   title, retrieving 211, 214  
   title, setting 512  
   visibility, determining 208  
   visibility, setting 447  
   Windows messages, sending 580  
 WINDOW ACTIVATE command 574  
 WINDOW ARRANGE command 575  
 WINDOW CLOSE command 576  
 WINDOW DEFAULT command 577  
 WINDOW function 209  
 window handle  
   of active child window 96  
   of active session window 107  
   of session window 115  
 window handles  
   default, setting 486  
   defined and use of 74  
   of active window, retrieving 96  
   of default session, retrieving 115

WINDOW HIDE command 578  
   example of 74  
 WINDOWHND function 210  
   example of 74  
 WINDOW MAXIMIZE command 579  
 WINDOW MESSAGE command 580  
 WINDOW MINIMIZE command 581  
 WINDOW MOVE command 582  
 WINDOWNAME function 211  
 WINDOW OPEN command 583–584  
 window operations, grouped 239  
 WINDOW RESTORE command 585  
 windows  
   session 74  
   window handles 74, 75  
 Windows messages, sending 580  
 Windows metafile graphics  
   adding to scripted dialog boxes 311  
   copying to a printer 325  
   copying to the clipboard 325  
 WINDOW STACK command 586  
 WINDOW UNHIDE command 587  
 WNDCLASS function 212  
 WNDFILE function 213  
 WNDTITLE function 214  
 word wrapping for file transfers 513

## X

XCLOCK messages 514  
 XFERCONFIG command 588–592  
 XModem configuration. *See* file transfer configuration.  
 XSYSTEM messages 515

## Y

YModem configuration. *See* file transfer configuration.

## Z

ZModem configuration. *See* file transfer configuration.  
 ZOOMED function 215